# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

Moderni real-time sistemi su zasnovani na komplementarnim konceptima multitaskinga i komunikacije između taskova (Intertask communications). Multitasking okruženje dozvoljava real-time aplikacijama da budu konstruisane kao skup nezavisnih taskova, svaki task sa svojom vlastitom niti izvršavanja i skupom sistemskih resursa. Komunikacija između taskova omogućava taskovima da se sinhronizuju i komuniciraju sa ciljem da koordiniraju njihove aktivnosti. Kod VxWorks operativnog sistema, mehanizmi za komunikaciju između taskova se kreću u rasponu od brzih semafora do redova poruka (Message queues) i od cjevi (Pipes) do mrežnih utičnica ( Sockets).

Još jedan ključni mehanizam kod real-time sistema je hardversko rukovanje interaptima, iz razloga što su interapti

obićno mehanizam za obavještavanje sistema o vanjskim događajima. Da bi dobili najbrži mogući odgovor za interapte, *intrrupt service routines (ISRs)* kod WxWorks OS-a izvršavaju se u posebnom kontekstu, izvan konteksta bilo kojeg taska.

**VxWorks taskovi**

Često je neophodno da se aplikacije organizuju u nezavisne, međusobno kooperativne programe. Svaki od ovih programa, dok se izvršava, zove se *task*.

***Multitasking***

Multitasking obezbjeđuje fundamentalni mehanizam jednoj aplikaciji da kontroliše i reaguje na višestruke, diskretne događaje iz realnog svijeta. VxWorks real-time kernel, *wind*, obezbjeđuje osnovno multitasking okruženje. Multitasking kreira mnoge niti koje se izvršavaju konkurentno, zapravo,

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

kernel pokreče njihovo izvršavanje na osnovu algoritma raspoređivanja. Svaki task ima svoj vlastiti kontekst, koji je CPU okruženje i sistemski resursi koje task vidi svaki put kad od strane kernela bude raspoređen da se izvršava. Kod izmjene konteksta, kontekst taska se sačuva u kontrolnom bloku taska (Task Control Block – TCB).

Kontekst taska uključuje:

- Izvršavanje niti; ( task programski brojač )
- CPU registri i floating-point registri (opciono)
- Stek za dinamičke varijable i pozive funkcija
- I/O za standardni ulaz, izlaz i grešku
- Tajmer kašnjenja
- Tajmer vremena
- Kontrolne strukture kernela

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

- Rukovatelji ( hendleri) signalima
- Vrijednosti za debagiranje i monitoring performansi

Kod VxWorksa, jedan bitan resurs koji nije dio konteksta taskova je memorijski adresni prostor: sav kod se izvršava u jednom adresnom prostoru. Dajući svakom tasku njegov vlastiti adresni prostor on zahtijeva virtualno-u-fizičko preslikavanje memorije, što je dostupno samo kod opcionog proizvoda VxVMI.

***Prelazna stanja taska***

Kernel održava tekuče stanje svakog taska u sistemu. Task se mijenja iz jednog stanja u drugo kao rezultat poziva funkcija kernela koje napravi aplikacija. Kada je kreiran, task ulazi u *suspended* stanje. Aktivacija je neophodna za task koji je kreiran da uđe u *ready* stanje. Faza aktivacije je izuzetno brza i omogućava aplikacijama da najprije keriraju taskove i da ih zatim na vrijeme aktiviraju.

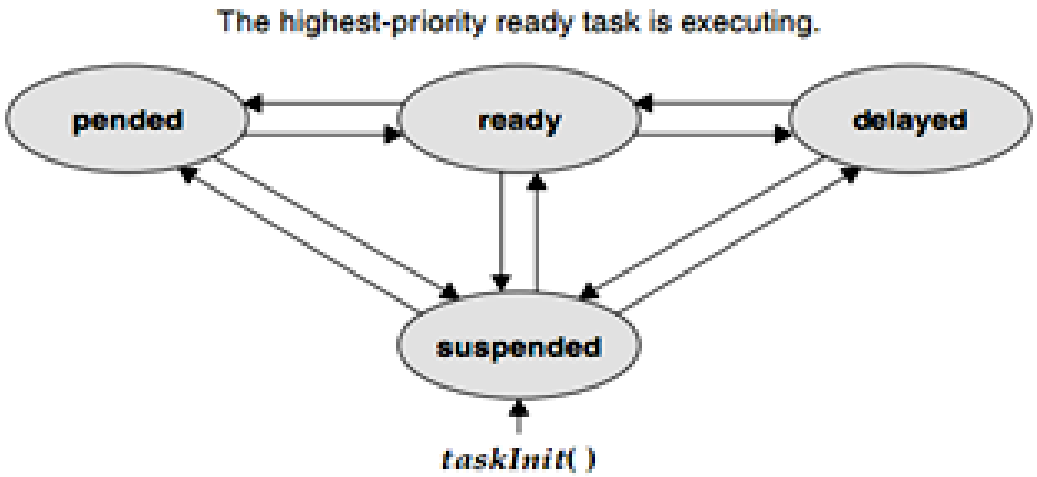
# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

Alternativa je *spawing*, koja dopušta tasku da bude kreiran i aktiviran sa samo jednom funkcijom. Taskovi mogu biti izbrisani iz bilo kojeg stanja. U narednoj tabeli su prikazana prelazna stanja taska.

| Simbol stanja | Opis |
|---|---|
| READY | Stanje taska koji ne čeka za bilo kakav drugi resurs osim procesora |
| PEND | Stanje taska koji je blokiran zbog nedostupnosti nekog resursa. |
| DELAY | Stanje taska koji spava neko vrijeme. |
| SUSPEND | Stanje taska koji nije dostupan za izvršenje. Ovo stanje se koristi isključivo za debagiranje. Suspenzija ne sprečava tranziciju taska, već samo njegovo izvršavanje. Dakle, pended-suspended taskovi još uvijek mogu deblokirani i delayed-suspended taskovi mogu biti probuđeni. |
| DELAY + S | Task je istovremeno u stanju suspended i delayed |
| PEND + S | Task je istovremeno u stanju pended i suspended |

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

| | |
|---|---|
| **PEND + T** | Task je u stanju pended sa nekom vrijednošću tajm-auta. |
| **PEND + S + T** | Task je u stanju pended sa vrijednošću tajmč-auta i u stanju suspended. |
| ***stanje* + I** | Stanje taska specificirano sa *stanje*, plus naslijeđeni prioritet. |




| | | | |
|---|---|---|---|
| ready | → | pended | semTake( ) / msgQReceive( ) |
| ready | → | delayed | taskDelay( ) |
| ready | → | suspended | taskSuspend( ) |
| pended | → | ready | semGive( ) / msgQSend( ) |
| pended | → | suspended | taskSuspend( ) |
| delayed | → | ready | expired delay |
| delayed | → | suspended | taskSuspend( ) |
| suspended | → | ready | taskResume( ) / taskActivate( ) |
| suspended | → | pended | taskResume( ) |
| suspended | → | delayed | taskResume( ) |

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

## *Wind raspoređivanje taska*

Multitasking zahtijeva algoritam raspoređivanja kako bi se spremnim taskovima dodijelio procesor. Defaultni algoritam kod winda je na prioritetu bazirano preiemptivno raspoređivanje. Također možemo odabrati da koristimo round-robin raspoređivanje za svoju aplikaciju. Obadva algoritma se oslanjaju na prioritet taskova. Wind kernel ima 256 nivoa prioriteta, numerisanih od 0 do 255. Prioritet 0 je najviši dok je 255 najniži prioritet. Prioritet se taskovima dodijeljuje kada se kreiraju. Nivo prioriteta taska se može promijeniti tokom njegovog izvršavanja pozivanjem **taskPrioritySet()** funkcije.

Rutine koje kontrolišu raspoređivanje taska prikazane su u narednoj tabeli:

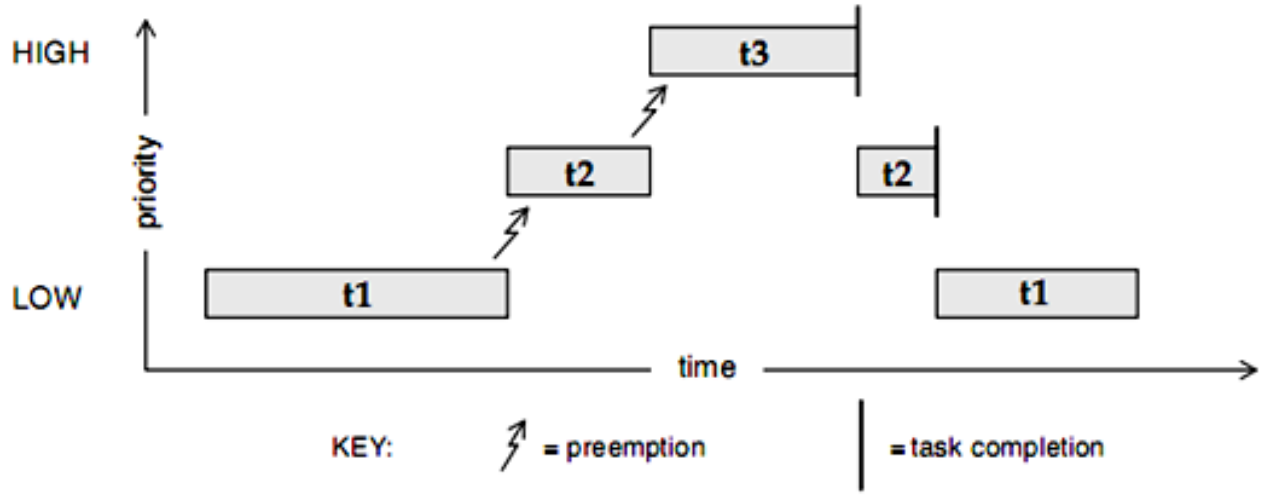
# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

| Poziv | Opis |
|---|---|
| kernelTimeSlice() | Kontroliše round-robin raspoređivanje |
| taskPrioritySet() | Mijenja priortiet tasku |
| taskLock() | Onemogućuje ponovno raspoređivanje taska |
| taskUnlock() | Omogućuje ponovno raspoređivanje taksa. |

## Priemptivno prioritetno rasporedjivanje

Priemptivni raspoređivač zasnovan na prioritetu, priemptira CPU kada task ima veći prioritet od taska koji se trenutno izvršava. Stoga, kernel osigurava da CPU bude uvijek dodijeljen tasku najvišeg prioriteta koji se već izvršava. Ovo znači da ako task sa višim prioritetom nego tekući task postane spreman za izvršavanje, kernel odmah skida kontekst trenutnog taska i mijenja ga kontekstom taska većeg prioriteta..

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM




Na gornjoj slici je task t1 priemptiran od taska većeg prioriteta t2 , koji je zatim priemptiran od taska t3. Kada se završi t3, nastavlja se izvršavanje t2 itd.

Nedostatak ovog algoritma raspoređivanja je da, kada više taskova istog prioriteta treba da dijeli procesor, ako jedan task nikada nije blokiran, on može da uzurpira procesor. Dakle, drugi taskovi istog prioriteta nikada neće dobiti šansu da se izvrše. Round-robin raspoređivanje riješava ovaj problem.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

## Round-robin raspoređivanje

Round-robin algoritam raspoređivanja pokušava da pravično dijeli CPU između svih spremnih taskova *istog prioriteta*. Round-robin raspoređivanje koristi *iscjekano vrijeme* (Time slicing) kako bi postigao fer dodijeljivanje procesora svim taskovima sa istim prioritetom. Svaki task, u grupi taskova sa istim prioritetom, izvršava se u definisanom intervalu odsječka vremena.

Round-robin raspoređivanje se uključuje pozivanjem **kernelTimeSlice()** funkcije, koja uzima parametar za odsječak vremena ili interval. Ovaj interval je količina vremena koja je dozvoljena svakom tasku da se izvrši prije nego što se procesor dodijeli drugom tasku istog prioriteta. Dakle, taskovi se rotiraju, pri čemu se svaki izvršava za jedan jednaki interval vremena. Nijedan task neče dobiti novi odsječak vremena prije nego što svim ostalim taskovima istog prioriteta unutar

grupe ne bude omogućeno da se izvršavaju.

Ako je round-robin raspoređivanje uključeno, i priempcija je uključena za task koji se izvršava, sistemski upravljač otkucajem povećava brojač vremena za task. Kada istekne specificirani odsječak vremena, sistemski upravljač sa novim otkucajem resetuje brojač i task se stavlja na začelje liste taskova prema njihovom prioritetu. Novi taskovi se pridružuju datoj grupi prioriteta i stavljaju se na začelje grupe kada njihov brojač bude inicijaliziran na nulu.

Uključivanje round-robin raspoređivanja ne utiče na performanse izmjene konteksta taska, niti se alocira dodatna memorija.

Ako je task blokiran ili priemptiran od strane taska većeg prioriteta za vrijeme njegovog intervala vremena, njegov vremenski brojač se zaustavlja a zatim se nastavlja kada task dođe na red za izvršavanje.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

U slučaju priemptiranja, task će nastaviti sa izvršavanjem kada se task većeg prioriteta izvrši, i kada se ustanovi da nema taska većeg prioriteta koji je spreman na izvršavanje. U slučaju kada je task blokiran, on se prebacuje na kraj liste taskova istog prioriteta. Ako je priemptiranje isključeno za vrijeme round-robin raspoređivanja, vremenski brojač taska koji se izvršava se ne inkrementira.

Vremenski brojači se obračunavaju po tasku koji se izvršava kada se desi sistemski otkucaj, bez obzira da li se ili ne task izvršava za cijeli interval otkucaja. Usljed priemptiranja od strane taska većeg prioriteta ili krađom CPU vremena dodijeljenog tasku od strane ISR-a, postoji mogućnost za task da se efektivno izvrši za više ili manje od ukupnog CPU vremena koje je dodijeljeno u trajanju odsječka vremena.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

## Zaključavanje priempcije (Preemption locks)

Wind raspoređivač može biti eksplicitno isključen ili uključen za pojedinačni task sa rutinama **taskLock()** i **taskUnlock()**. Kada task isključi raspoređivač pozivom funkcije **taskLock()**, izmještanje koje nije zasnovano na prioritetu može preuzeti posao dok se task izvršava.

Međutim, ako se task eksplicitno blokira ili suspendira, raspoređivač odabire slijedeći task sa najvećim prioritetom. Kada se task sa blokiranom priempcijom deblokira i počne ponovo sa izvršavanjem, priempcija se ponovo isključuje.

Treba zapamtiti da zaključavanje priemptiranja spriječava izmjenu konteksta taska, ali ne blokira rukovanje interaptima. Zaključavanje priemptiranja se može koristiti da se postigne uzajamno isključivanje; ipak, trajanje zaključavanja treba svesti na minimum.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

## *Kontrola taska*

U nastavku slijedi pregled nekih osnovnih VxWorks rutina koje se mogu pronaći u Vxworks biblioteci **taskLib**. Ove rutine su sredstvo za kreiranje taska, kontrolu nad taskom kao i za dobijanje informacija o taskovima.

### Kreiranje i aktivacija taska

Rutine prikazane u narednoj tabeli se koriste za kreiranje taska.

| Poziv | Opis |
|---|---|
| taskSpawn() | Kreira i aktivira novi task |
| taskInit() | Inicijalizira novi task |
| taskActivate() | Aktivira inicijalizirani task |

Argumenti za **taskSpawn()** su naziv novog taska (ASCII string), prioritet taska, „options“ riječ, veličina steka, adresa glavne rutine i 10 argumenata koji mogu biti proslijeđeni glavnoj rutini kao početni parametri:

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

**Id = taskSpawn(name, priority, options, stacksize, main, arg1, ..., arg10);**

Ova rutina kreira kontekst novog taska koji uključuje alokaciju steka i postavljanje okruženja taska za poziv glavne rutine sa specificiranim argumentima. Novi task počinje izvršavanje na početku specificirane rutine.

Rutina **taskSpawn()** utjelovljuje korake nižeg nivoa alokacije, inicijalizacije i aktivacije. Funkcije inicijalizacije i aktivacije su obezbjeđene od strane rutina **taskInit()** i **taskActivate()**.

**Stek taska**

Teško je znati koliko prostora na steku je potrebno alocirati. Da bi izbjegli stack overflow, i stack corruption od strane taska, može se slijediti slijedeći pristup; Kada inicijalno alociramo stek, trebamo ga načiniti što je moguće većim nego što je predviđeno; na primjer od 20KB na 100KB, u zavisnosti od vrste aplikacije.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

Zatim je potrebno periodično nadgledati stek sa **checkStack()**, i ako je dovoljno veliko, modifikovanjem veličine možemo ga načiniti manjim.

**Naziv taska i identifikacijski broj**

Kada je task kreiran i aktiviran, može se specificirati ASCII string bilo koje dužine koji će predstavljati naziv taska. VxWorks vraća ID taska koji je 4-bajtni pokazivač na strukture podataka taska. Većina VxWorks rutina za rad sa taskovima uzima ID taska kao argument. VxWorks koristi konvenciju prema kojoj task ID vrijednosti 0 (nula) uvijek podrazumijeva pozivajući task. VxWorks ne zahtijeva da imena taskova budu jedinstvena, ali se preporučuje da se koriste jedinstvena imena kako bi se izbjeglo zbunjivanje korisnika. Da bi se izbjegli konflikti VxWorks koristi konvenciju za označavanje svih naziva taskova sa target mašine sa slovom *t* i naziva taskova koji se pokreću sa hosta sa slovom *u*.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

Nekada nećemo željeti da dajemo nazive nekim ili svim taskovima aplikacije. Ako je NULL, pokazivač služi umjesto name argumenta od **taskSpawn()**, tada VxWorks dodijeljuje jedinstven naziv. Naziv je u obliku *tN* gdje je N decimalni integer koji je inkrementiran za jedan za svaki neimenovani task koji je kreiran i aktiviran.

Rutine taskLib koje su prikazane u narednoj tabeli upravljaju ID-jevima i nazivima taska.

| Poziv | Opis |
|---|---|
| taskName() | Dohvaća naziv taska koji je asociran sa ID-jem taska. |
| taskNameTold() | Dohvaća Id taska koji je asociran za nazivom taska. |
| taskIdSelf() | Dohvaća ID pozivajučeg taska |
| taskIdVerify() | Potvrđuje postojanje specificiranog taska |

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

## Opcije taska

Kada je task kreiran i aktiviran, možemo proslijediti jedan ili više opcionih parametara koji su prikazani u slijedećoj tabeli. Rezultat je određen obavljanjem logičke ILI operacije nad specifičnom opcijom.

| Naziv | Hex vrijednost | Opis |
|---|---|---|
| VX_FP_TASK | 0x0008 | Izvršavanje sa floating-point koprocesorom |
| VX_NO_STACK_FILL | 0x0100 | Ne popunjava stek sa 0xee |
| VX_PRIVATE_ENV | 0x0080 | Izvršava task sa privatnim okruženjem |
| VX_UNBREAKABLE | 0x0002 | Onemogučava breakpointe za task |
| VX_DSP_TASK | 0x0200 | 1 = DSP podrška koprocesora |
| VX_ALTIVEC_TASK | 0x0400 | 1 = ALTIVEC podrška koprocesora |

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

Treba uključiti VX_FP_TASK opciju kada kreiramo task koji:

- Obavlja operacije u pokretnom zarezu
- Poziva bilo koju funkciju koja vraća floating-point vrijednost
- Poziva bilo koju funkciju koja uzima floating-point vrijednost kao argument.

Na primjer:

**tid = taskSpawn(„tMojTask“, 90, VX_FP_TASK, 20000, myFunc, 2387, 0, 0, 0, 0, 0, 0, 0, 0, 0);**

Neke rutine obavljaju operacije sa pokretnim zarezom interno. VxWorks dokumentacija za svaku od ovih rutina jasno naglašava potrebu za korištenjem VX_FP_TASK opcije.

Nakon što je task kreiran i aktiviran možemo otkriti ili promijeniti opcije taska korištenjem rutina prikazanih u slijedećoj tabeli:

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

| Poziv | Opis |
|---|---|
| taskOptionGet() | Otkriva opcije taska |
| taskOptionsSet() | Postavlja opcija taska |

## Informacije o tasku

Rutine prikazane u narednoj tabeli daju informacije o tasku uzimanjem snimka konteksta taska kada je rutina pozvana. Obzirom da je stanje taska dinamičko, informacija možda neće biti najsvježija sve dok task ne bude suspendiran

:

| Poziv | Opis |
|---|---|
| taskIdListGet() | Popunjava niz sa ID-jevima svih aktivnih taskova. |
| taskInfoGet() | Dohvaća informacije o tasku |
| taskPriorityGet() | Ispituje prioritet taska |
| taskRegsGet() | Ispituje registre taskova (ne može se koristiti sa aktuelnim taskom) |
| taskRegsSet() | Postavlja registre taskova (ne može se koristiti sa aktuelnim taskom) |
| taskIsSuspended() | Provjerava da li je task suspendiran |
| taskIsReady() | Provjerava da li je task spreman za izvršavanje |
| taskTcb() | Dohvaća pokazivač na kontrolni blok taska |

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

## Brisanje taska i sigurnost brisanja

Taskovi mogu biti dinamički obrisani iz sistema. VxWorks sadrži rutine prikazane u narednoj tabeli za brisanje taskova i zaštitu taskova od neočekivanog brisanja.

| Poziv | Opis |
|---|---|
| exit() | Prekida pozivajući task i oslobađa memoriju. |
| taskDelete() | Prekida specificirani task i oslobađa memoriju |
| taskSafe() | Štiti pozivajući task od brisanja |
| taskUnsafe() | Poništava taskSafe() |

Task može ubiti drugi task ili samog sebe pozivanjem **taskDelete()**. Kada je task obrisan, nijedan drugi task nije obavjišten o njegovom brisanju. Rutine **taskSafe()** i **taskUnsafe()** riješavaju probleme koji proizilaze iz neočekivanog brisanja taskova. Rutina **taskSafe()** štiti task od brisanja od strane drugog taska. Ova zaštita je često potrebna kada se task izvršava u kritičnom regionu ili pristupa kritičnom

resursu.

Na primjer, task može uzeti semafor za ekskluzivan pristup nekim strukturama podataka. Dok se izvršava unutar kritičnog regiona, task može biti obrisan od strane drugog taska. Pošto je task u nemogućnosti da kompletira kritični region, struktura podataka može biti ostavljena u zauzetom ili nekonzistentnom stanju. Štaviše, kako semafor ne može biti oslobođen od strane taska, kritični resurs je sada nedostupan za korištenje od bilo kojeg drugog taska i on je u suštini zamrznut.

Bilo koji task koji pokušava da obriše task koji je zaštićen sa sa **taskSafe()** je blokiran. Kada završi sa svojim kritičnim resursom, zaštičeni task može sebe učiniti dostupnim za brisanje pozivanjem **taskUnsafe()**, koji sprema bilo koji task za brisanje. Da bi podržao ugnježdene regione zaštičene od brisanja vodi se računa o broju poziva **taskSafe()** i **taskUnsafe()** rutina. Brisanje je dozvoljeno samo kada je broj

jednak nuli.

Samo je pozivajuči task zaštićen. Task ne može učinit drugi task sigurnim ili nesigurnim od brisanja.

Slijedeći fragment koda prikazuje kako koristiti taskSafe() i taskUnsafe() da bi zaštitili kritični region koda:

```
taskSafe();
semTake(semId, WAIT_FOREVER); /*Blokiraj dok je semafor dostupan */
.
.   /* kritični region koda */
.
semGive(semId);                    /* Oslobodi semafor */
taskUnsafe();
```

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

Sigurno brisanje je obično usko povezano sa međusobnim isključivanjem, kao u ovom primjeru. Zbog pogodnosti i efikasnosti, poseban oblik semafora, mutual-exclusion semafor, nudi opciju za sigurno brisanje.

## Kontrola taska

Rutine prikazane u narednoj tabeli pružaju direktnu kontrolu nad izvršavanjem taska. VxWorks debugiranje zahtijeva rutine za zaustavljanje i ponovno pokretanje taska. One se koriste da zamrznu stanje taskova za ispitivanje

| Poziv | Opis |
|---|---|
| taskSuspend() | Suspenduje task |
| taskResume() | Nastavlja sa izvršavanjem taska |
| taskRestart() | Ponovo pokreće task |
| taskDelay() | Odgađa task; jedinice odgađanja su tikovi, rezolucija u tikovima |
| Nanosleep() | Odgađa task; jedinice odgađanja su nanosekunde, rezolucija u tikovima |

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

Taskovi će možda zahtijevati ponovno pokretanje (restart) tokom izvršavanja sa ciljem da se odgovori na neke katastrofične greške. Mehanizam restarta, **taskRestart()**, ponovo kreira task sa originalnim argumentima.

Operacije odgađanja pružaju jednostavan mehanizam za task da ide na spavanje za fiksno trajanje. Odgađanje taska se obično koristi za polling aplikacije. Na primjer, da odgodimo task na pola sekunde bez ikakvih pretpostavki o taktu, poziv je:

**taskDelay(sysClkRateGet() / 2);**

Rutina **sysClkRateGet()** vraća brzinu sistemskog sata u tikovima po sekundi. Umjesto taskDelay(), može se koristiti POSIX rutina **nanosleep()** da se specificira odgađanje direktno u vremenskim jedinicama. Samo su jedinice različite; rezolucija kod obadvije rutine je ista, i zavisi od sistemskog sata.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

Kao popratni efekat, **taskDelay()** premješta pozivajući task na kraj reda spremnih taskova istog prioriteta. Praktično, možemo dati procesor bilo kojem tasku istog prioriteta „odgađanjem“ za nula tikova sata:

**taskDelay(NO_WAIT); /*dozvoli drugim taskovima istog prioriteta da se pokrenu */**

„Odgađanje“ od nula tikova trajanja je moguće samo sa **taskDelay()**; **nanosleep()** ga smatra greškom.

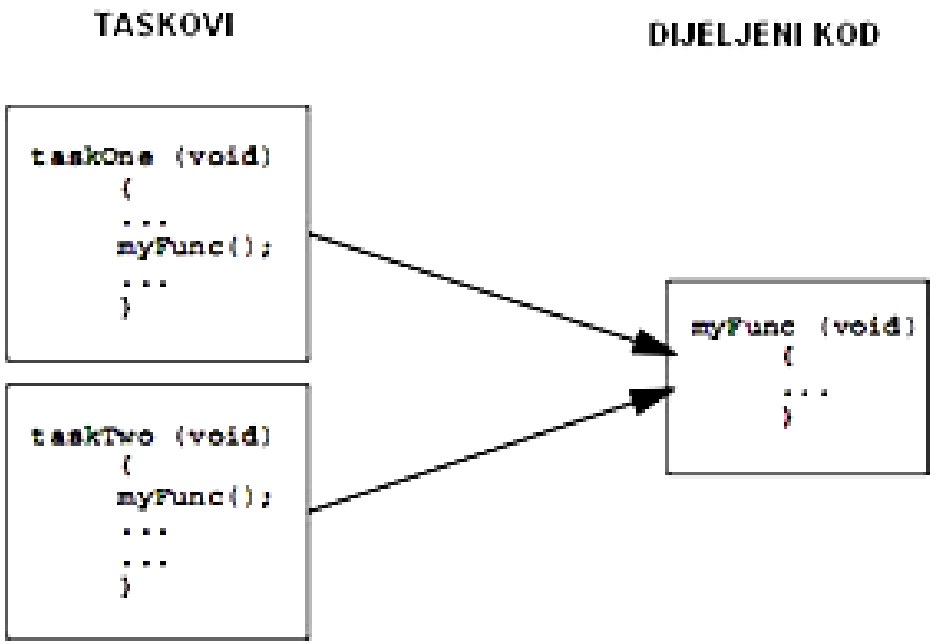
***Dijeljeni kod i reentrant kod***

U VxWorksu, uobičajeno je za jednu kopiju subrutine ili subrutine iz biblioteke, da bude pozvana od strane više različitih taskova. Na primjer, mnogi taskovi mogu pozivati **printf()**, ali postoji samo jedna kopija ove subrutine u sistemu. Jedna kopija koda pozivana od strane više taskova se zove

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

dijeljeni kod.

Dinamičke osobine VxWorksa čine ovo posebno jednostavnim. Dijeljeni kod čini sistem efikasnijim i lakšim za održavanje; kao što jse vidi sa slijedeće slike



# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

Dijeljeni kod mora biti oslobođen konflikta - reentrant. Subrutina je oslođena konflikta ako jedna kopija rutine može biti simultano pozvana iz nekoliko konteksta taskova bez konflikta. Konflikt se obično pojavi kada subrutina mijenja globalne ili statičke varijable, jer postoji samo jedna kopija podatka i koda. Većina rutina kod VxWorks operativnog sistema su reentrant rutine. Međutim, treba uzeti u obzir da bilo koja rutina **nekoIme()** nije reentrant rutina ako ima odgovarajuće rutine koja je imenovana kao **nekoIme_r()**. Na primjer, kako **ldiv()** ima odgovarajuću rutinu **ldiv_r()**, možemo zaključiti da **ldiv()** nije reentrant.

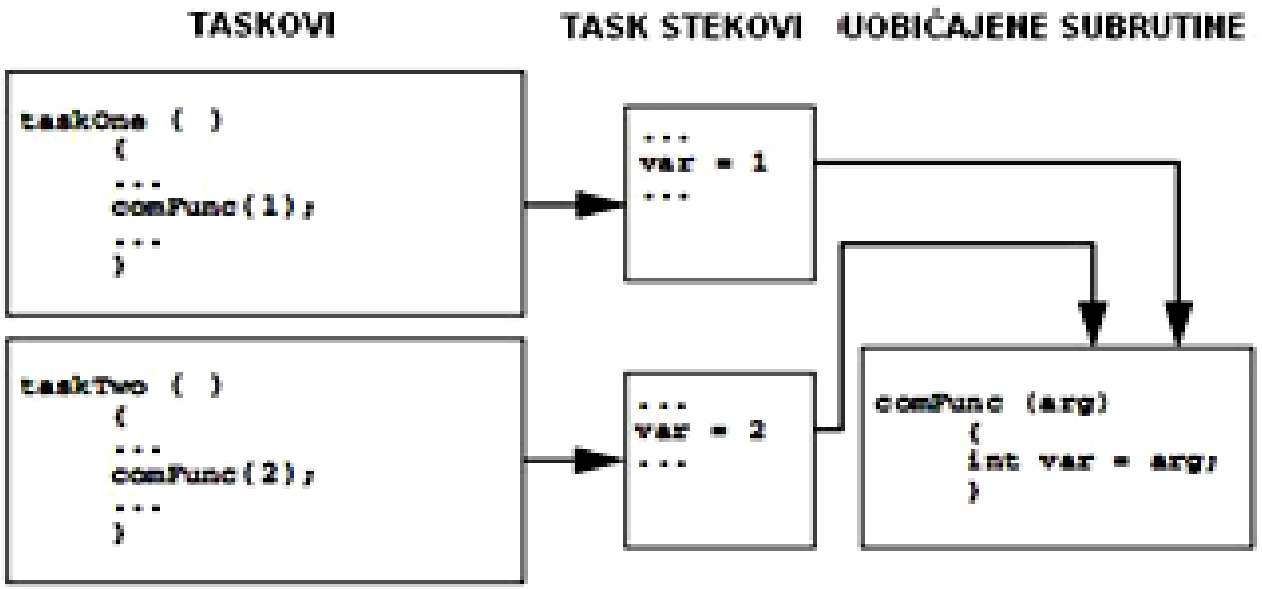
VXWorks I/O i rutine drajvera su reentrant, ali zahtijevaju pažljiv dizajn aplikacije. Za baferovani I/O preporučuje se korištenje fajl-pointer bafera po task osnovi. Na nivou drajvera, moguće je učitati bafere sa tokovima iz različitih taskova.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

## Varijable dinamičkog steka

Većina subrutina je čisti kod, nemaju vlastitih podataka izuzev dinamičkih stek varijabli. One rade ekskluzivno na podacima obezbijeđenim od strane pozivaoca kao parametri. Biblioteka vezanih listi, lstLib, je dobar primjer. Njihove rutine rade na listama i čvorovima koje obezbjeđuje pozivaoc pri svakom pozivu subrutine. Subrutine ove vrste su inherentno reentrant. Više taskova mogu koristiti ove rutine simultano, bez međusobne interferencije, iz razloga što svaki task ima svoj vlastiti stek. Vidjeti slijedeću sliku .

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM




## Čuvane globalne i statičke varijable

Neke biblioteke enkapsuliraju pristup zajedničkim podacima. Ova vrsta biblioteke zahtjeva neki oprez iz razloga što rutine nisu inherentno reentrant. Višestruki taskovi simultano pozivaju rutine u biblioteci i mogu interferirati sa pristupom zajedničkim podacima. Takve biblioteke moraju biti napravljene eksplicitno

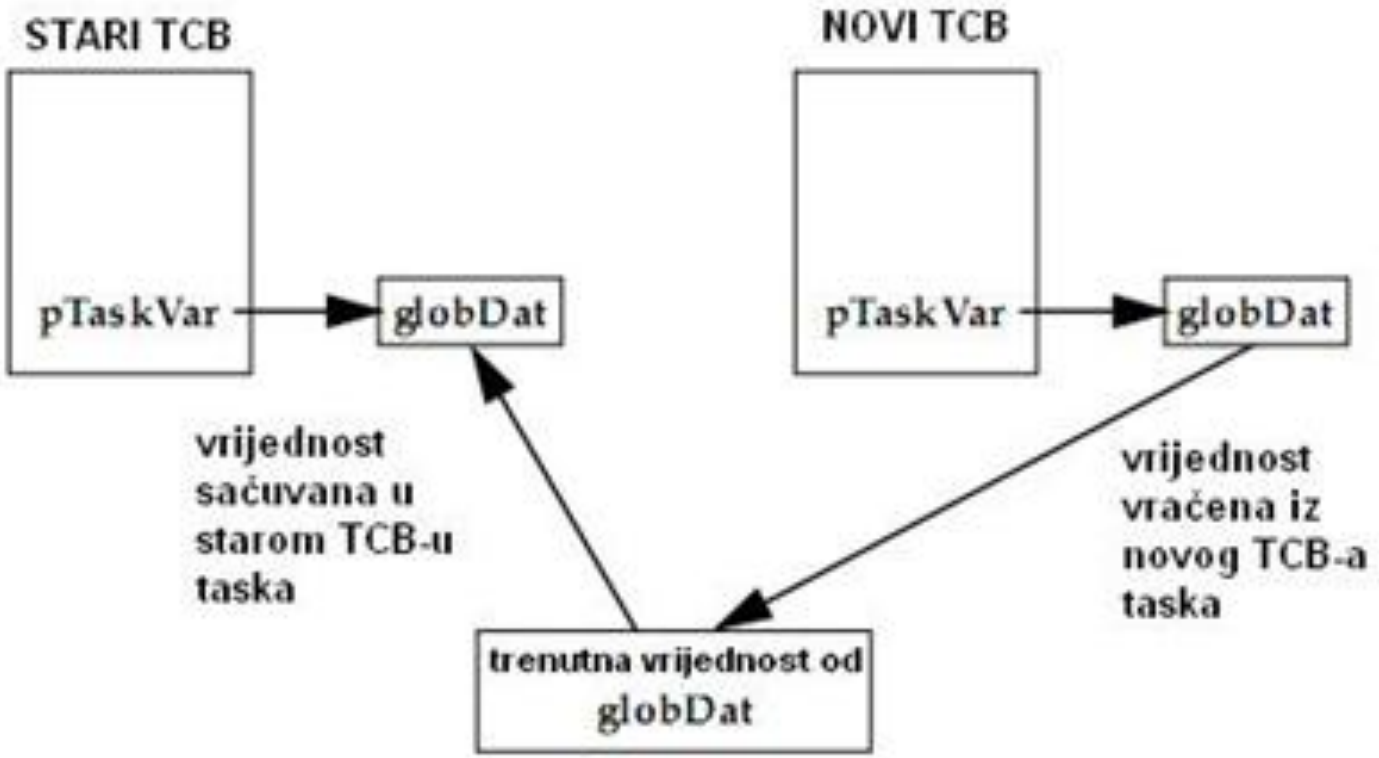
reentrantnim omogućavanjem mehanizma međusobnog isključivanja ( mutual-exclusion) da zabrani taskovima da simultano izvršavaju kritične sekcije koda. Uobičajeni mehanizam uzajamnog isključivanja je mutex semafor .

**Varijable taska**

Neke rutine koje se mogu simultano pozvati od strane više taskova mogu zahtijevati globalne ili statičke varijable sa različitim vrijednostima za svaki pozivajući task. Na primjer, nekoliko taskova može biti upućeno na privatni bafer a da se ipak odnosi na istu globalnu varijablu.

Da bi olakšao stvar, VxWorks pruža mogućnost nazvanu *varijable taska* koje dozvoljavaju 4-bajtnim varijablama da budu dodane kontekstu taskova, tako da je vrijednost takve varijable promijenjena svaki put kad se task promijeni ka ili od njegovog taska vlasnika. Obično, nekoliko taskova deklarišu istu varijablu (4-bajtna memorijska lokacija) kao varijablu

taska. Svaki od ovih taskova može tretirati jednu memorijsku lokaciju kao svoju privatnu varijablu; kao na slici. Ova osobina je omogućena sa rutinama **taskVarAdd()**, **taskVarDelete()**, **taskvarSet()** i **taskvarGet()**.



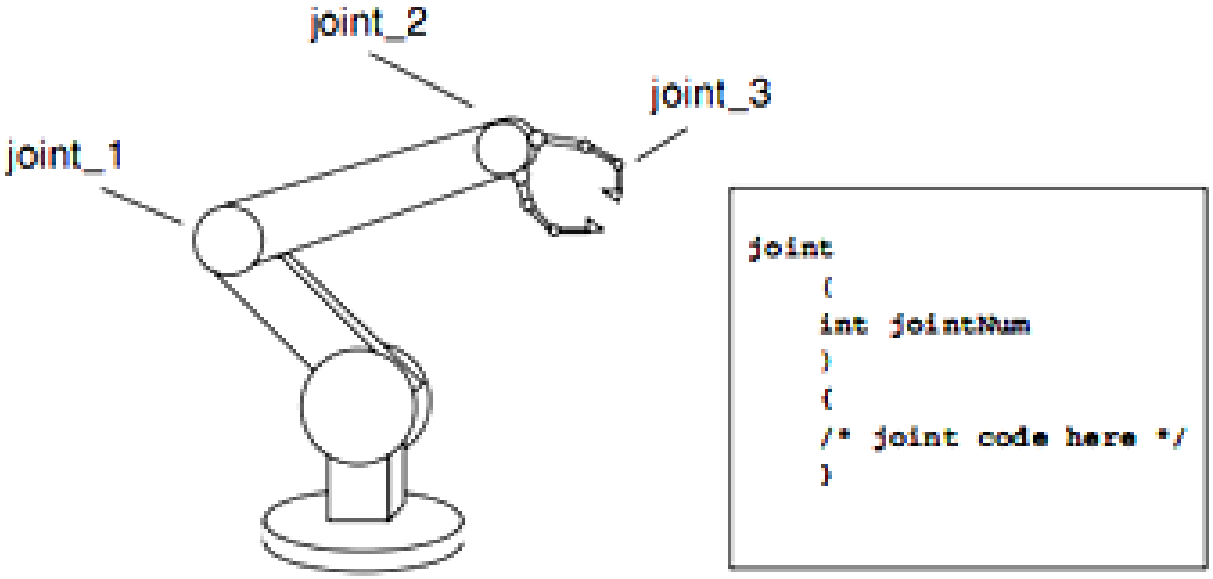
Ovaj mehanizam treba koristiti umjereno. Svaka varijabla taska dodaje nekoliko milisekundi na izmjenu konteksta za svoj task, iz razloga što vrijednost varijable mora biti sačuvana i ponovo vračena kao dio konteksta taska.

**Različiti taskovi sa istom Main rutinom**

Sa VxWorks operativnim sistemom moguće je kreirati i pokrenuti nekoliko taskova sa istom glavnom rutinom. Svako kreiranje i pokretanje kreira novi task sa svojim vlastitim stekom i kontekstom. Svako kreiranje i pokretanje također može proslijediti glavnoj rutini različite paramtere novom tasku. Ovo je korisno kada ista funkcija treba da bude pokrenuta konkurentno sa različitim skupom parametara. Na primjer, rutina koja nadgleda određenu vrstu opreme može biti kreirana i pokrenuta nekoliko puta kako bi nadgledala nekoliko različitih dijelova te opreme. Argumenti glavnoj rutini bi trebali ukazivati koji tačno dio opreme task nadgleda.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

Na narednoj slici , nekoliko dijelova mehaničke ruke koristi isti kod. Taskovi upravljaju dijelovima pozivanjem joint() procedure. Joint broj (jointNumber) se koristi da se označi kojim dijelom ruke se manipuliše.



# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

## Sistemski taskovi VxWorks operativnog sistema

U zavisnosti od konfiguracije, VxWorks sadrži različite sistemske taskove. Slijedi opis nekih od njih.

### *Root task: tUsrRoot*

Root task je prvi task koji se pokreće od strane kernela. Polazna tačka root taska je userRoot() iz installDir/**target/config/all/usrConfig.c** i inicijalizira većinu VxWorks sadržaja. On kreira i pokreće taskove kao što su task za prijavu na sistem, task za pokretanje, mrežni task i tRlogind deamon proces. Normalno, root task se terminira i briše nakon što se obavi cjelokupna inicijalizacija.

### *Task za logove: tLogTask*

Log task, tLogTask, se koristi od strane vxWorks modula da upisuje sistemske poruke bez potrebe za obavljanjem I/O operacija u tekućem kontekstu taska.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

***Task izuzetka: tExcTask***

Task izuzetka, tExcTask, podržava VxWorks paket za izuzetke, obavljanjem funkcija koje se ne mogu pojaviti na nivou interapta. Također se koristi za akcije koje ne mogu biti obavljene u trenutnom kontekstu taska, kao što je samoubistvo taska. Mora imati najveći prioritet u sistemu. Ne treba suspendovati, brisati ili mijenjati prioritet ovog taska.

***Mrežni task: tNetTask***

tNetTask demon rukuje funkcijama na nivou taska koje se zahtijevaju od VxWorks mreže.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

## Komunikacija između taskova

Komplement rutinama za multitasking su rutine za komunikaciju između taskova. Ove rutine dozvoljavaju da međusobno nezavisni taskovi koordiniraju njihove akcije.

WxVorks nudi bogat skup mehanizama za komunikaciju između taskova uključujući:

- Dijeljenu memoriju, za jednostavno dijeljenje podataka.
- Semafore, za osnovno međusobno isključivanje i sinhronizaciju.
- Mutexe i uslovne varijable za međusobno isključivanje i sinhronizaciju korištenjem POSIX interfejsa.
- Redove poruka i cijevi, za proslijeđivanje poruka između taskova unutar procesora.
- Utičnice (sokite) i udaljeno pozivanje procedura (RPC), za mrežnu komunikaciju između taskova.

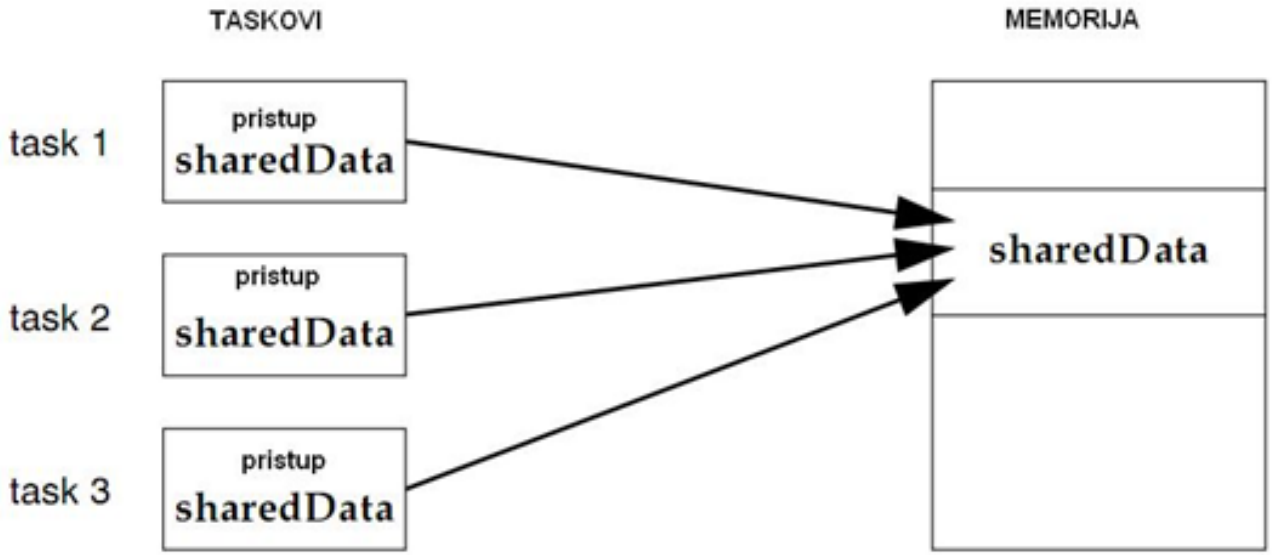
# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

- Signale, za rukovanje izuzecima.

***Dijeljene strukture podataka***

Najočitiji način za komunikaciju između taskova je pristupanjem strukturama dijeljenih podataka. Obzirom da svi taskovi u VxWorksu egzistiraju u jednom linearnom adresnom prostoru, dijeljene podataka između taskova je trivijalno; kao što je to prikazano na narednoj slici. Globalne varijable, linearni baferi, prstenasti baferi, vezane liste i pointeri mogu biti referencirani direktno od strane koda koji se izvršava u različitim kontekstima.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM




***Uzajamno isključivanje***

Dok dijeljeni adresni prostor pojednostavljuje razmjenu podataka, pristup memoriji blokiranjem je bitan za izbjegavanje konflikata. Razvijeno je dosta metoda za ostvarivanje ekskluzivnog pristupa resursima, i variraju samo u opsegu isključivanja. Takve metode uključuju isključivanje

interpata, isključivanje preempcije i zaključavanje resursa sa semaforima.

**Zaključavanje interapta i latencija**

Najmoćniji raspoloživi metod za međusobno isključivanje je onemogučavanje interapta. Ovakvo zaključavanje garantuje ekskluzivan pristup procesoru:

```
funcA ()
 {
  Int lock = intLock();
  .
  .   /*kritični region koda koji se ne može prekinuti */
  .
  intUnlock(lock);
  }
```

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

Dok ovo riješava probleme koji uključuju uzajamno isključivanje sa ISR-ama, ovo nije dovoljno kao metod uzajamnog isključivanja opšte namjene za većinu real-time sistema iz razloga što štiti sistem od određenih vanjskih događaja za vrijeme trajanja ovih zaključavanja. Latencija interapta je neprihvatljiva gdje god se zahtijeva hitan odgovor na vanjski događaj. Međutim, zaključavanje interapta ponekad može biti neophodno gdje uzajamno isključivanje uključuje ISR-ove. U bilo kojoj situaciji, treba držati zaključavanja interapta što je moguće kračim.

**Preemptivno zaključavanje i latencija**

Onemogučavanje preempcije nudi manje restriktivan oblik uzajamnog isključivanja. Dok nijednom drugom tasku nije dozvoljeno da skine task koji se trenutno izvršava, servisne rutine prekida su u mogućnosti da izvršavaju:

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

```
funcA ()
{
taskLock();
.
.     /*kritični region koda koji se ne može prekinuti */
 .
  taskUnlock(lock);
 }
```

Međutim, ovaj metod može dovesti do neprihvatljivog odgovora za real-time aplikaciju. Taskovi višeg prioriteta su u nemogućnosti da se izvrše sve dok task koji drži zaključanim kritični region ne napusti isti, čak iako task višeg prioriteta nije uključen sa kritičnim regionom. Dok je ovakav oblik uzajamnog isključivanja jednostavan, ako ga koristimo, trebamo ga koristiti za kratko vrijeme. Bolji mehanizam je

osiguran korištenjem semafora koji su objašnjeni u nastavku.

***Semafori***

VxWorks semafori su visoko optimizirani i pružaju najbrži mehanizam komunikacije između taskova u VxWorksu. Semafori su primarno sredstvo za zahtjeve uzajamnog isključivanja i sinhronizacije taskova, kao što je opisano ispod:

Za uzajamno isključivanje semafori blokiraju pristup dijeljenim resursima. Oni obezbjeđuju uzajamno isključivanje sa finijom granulacijom.

Za sinhronizaciju semafori koordiniraju izvršavanje taska sa vanjskim događajima.

Postoje tri vrste Wind semafora optimiziranih da podrže različite vrste problema:

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

- **Binarni (Binary)** – nabrži semafori opšte namjene. Optimizirani za sinhronizaciju ili uzajamno isključivanje.
- **Uzajamno isključivanje (Mutual exclusion)** – posebni binarni semafor optimiziran za probleme vezane za uzajamno isključivanje: nasljeđivanje prioriteta, sigurno brisanje i rekurzija.
- **Brojački (Counting)** – kao binarni semafor, ali drži informaciju o broju koliko je puta semafor predat. Optimiziran za čuvanje nekoliko instanci resursa.

  VxWorks ne nudi samo Wind semafor, dizajnirane isključivo za Vxworks, već također nudi POSIX semafore, dizajnirane za portabilnost.
- **Čitanje/Pisanje** – specijalna vrsta semafora koja osigurava medjusobno isključivanje za taskove koji trebaju pravo pisanja nad objektom, i konkurentni pristup za taskove koji samo trebaju pravo čitanja nad objektom.( korisno kod SMP)

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

## Upravljanje semaforima

Umjesto da se definiše pun skup rutina za upravljanje semaforima za svaku vrstu semafora, Wind semafori pružaju jedan uniforman interfejs za upravljanje semaforima. Samo su rutine za kreiranje specifične za svaku vrstu semafora. Naredna tabela prikazuje rutine za upravljanje semaforima.

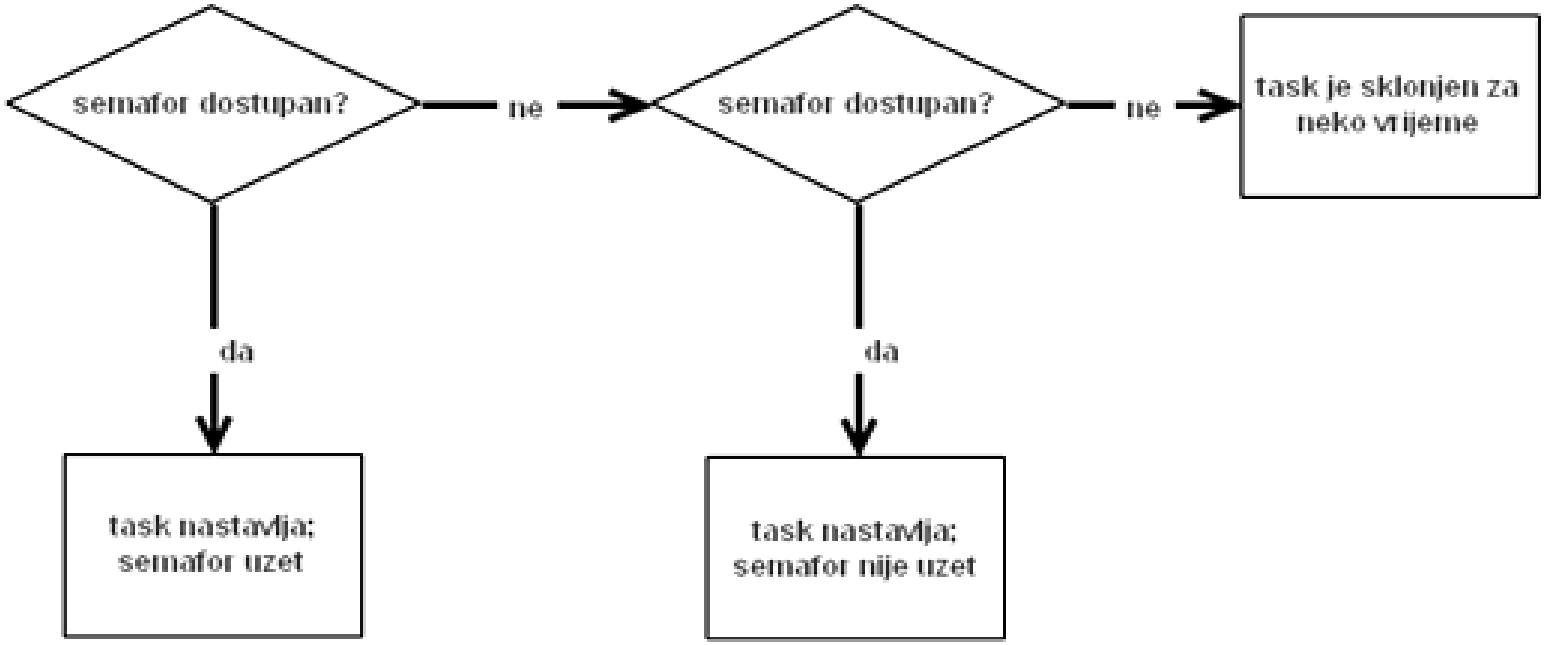
| Poziv | Opis |
|---|---|
| semBCreate() | Alocira i inicijalizira binarni semafor. |
| semMCreate() | Alocira i inicijalizira semafor uzajamnog isključivanja |
| semCCreate() | Alocira i inicijalizira brojački semafor. |
| semDelete() | Terminira i oslobađa semafor. |
| semTake() | Uzima semafor. |
| semGive() | Daje semafor. |
| semFlush() | Deblokira sve taskove koji čekaju na semafor. |

Rutine **semBCreate()**, **semMCreate()** i **semCreate()** vračaju ID semafora koji služi za rukovanje semaforom tokom njegovog korištenja od strane druge rutine za kontrolu semaforima.

**Binarni semafori**

Binarni semafor opšte namjene je u mogućnosti da zadovolji zahtjevima uzajamnog isključivanja i sinhronizacije. Binarni semafor se može posmatrati kao zastavica koja je dostupna (full) ili nedostupna (empty). Kada task uzme binarni semafor, sa **semTake()**, rezultat će da zavisi od toga da li je semafor dostupan (full) ili nedostupan (empty) u trenutku poziva; vidjeti narednu sliku.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM




Ako je semafor dostupan (full), tada semafor postaje nedostupan (empty) i task trenutno nastavlja da se izvršava. Ako je semafor nedostupan (empty), task se stavlja u red blokiranih taskova i ulazi u stanje čekanja na dostupnost semafora. Kada task preda binarni semafor, korištenjem **semGive()**, ishod također zavisi toga da li je semafor

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

dostupan (full) ili nedostupan (empty) u trenutku poziva; vidjeti na slijedećoj slici. Ako je semafor već dostupan (full), davanje semafora nema efekta. Ako je semafor nedostupan (empty) i nema taska koji ćeka da ga uzme, tada semafor postaje dostupan (full). Ako je semafor nedostupan (empty) i jedan ili više taskova čekaju na njegovu dostupnost, tada se prvi task u redu blokiranih taskova odblokira i semafor postaje nedostupan (empty).

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

## Uzajamno isključivanje

Binarni semafori efikasno zaključavaju pristup dijeljenom resursu. Umjesto onemogučavanja interapta ili priemptivnih brava, binarni semafori ograničavaju opseg uzajamnog isključivanja na samo vezani resurs. Kod ove tehnike, semafor je kreiran da štiti resurs. Inicijalno je semafor dostupan (full).

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

```
 /*includes*/
„include „vxWorks.h“
 #include „semLib.h“
SEM_ID semMutex;
/*Kreiraj binarni semafor koji je inicijalno full. Taskovi *
*blokirani na semaforu čekaju po redoslijedu prioriteta    */
 semMutex = semBCreate(SEM_Q_PRIORITY, SEM_FULL);
```

Kada task želi da pristupi resursu, mora najprije da uzme semafor. Sve dok task drži semafor, svi drugi taskovi koji pokušavaju da pristupe resursu su blokirani. Kada task završi sa resursom, on vraća nazad semafor dozvoljavajući drugom tasku da koristi semafor.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

Prema tome, svi pristupi resursu zahtijevaju uzajamno isključivanje su ograđeni sa **semTake()** i **semGive()** parom:

```
semTake(semMutex, WAIT_FOREVER);
.
. /*Kritični region kojem može pristupiti samo jedan task u jednom trenutku */
.
semGive(semMutex);
```

## Semafori uzajamnog isključivanja

Semafor uzajamnog isključivanja je specijalizirani binarni semafor dizajniran da riješi pitanja uzajamnog isključivanja, uključujući inverziju prioriteta, sigurno brisanje i rekurzivni pristup resursima.

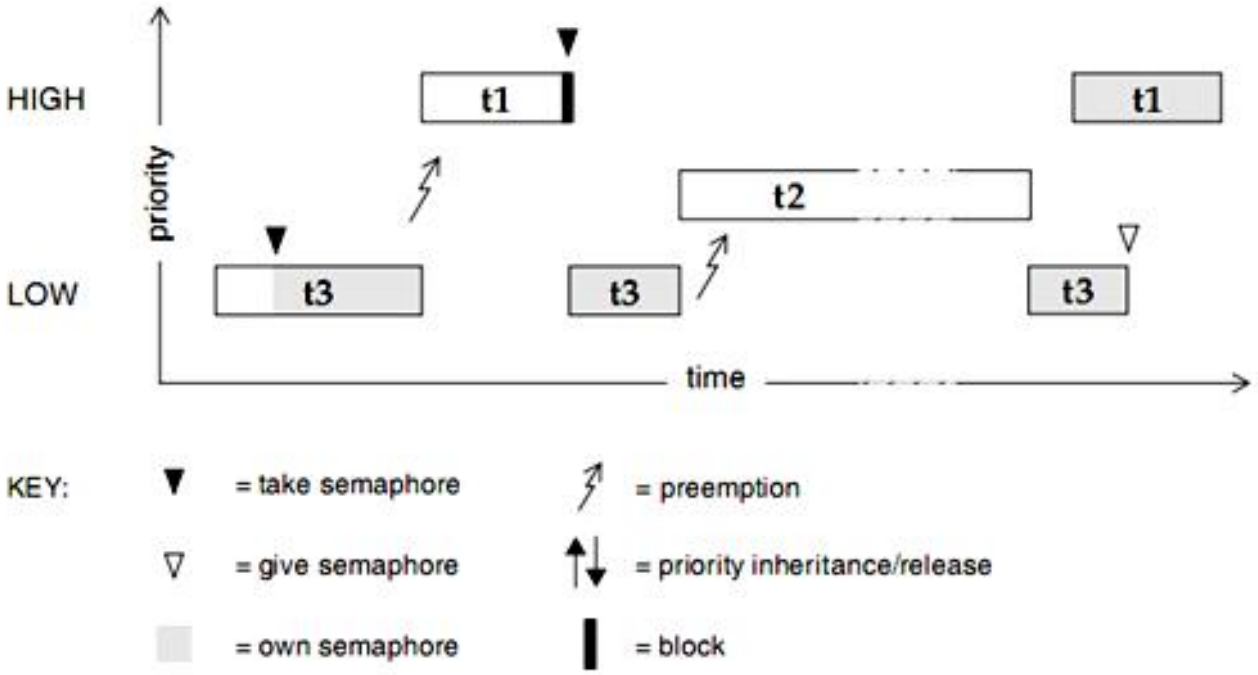
Osnovno ponašanje ove vrste semafora je identično binarnom semaforu uz slijedeće izuzetke:

- Može se koristiti za uzajamno isključivanje.
- Može biti dat samo onom tasku koji ga uzme.
- Ne može biti dat iz ISR.
- Operacija **semFlush()** nije dozvoljena.

***Prioritetna inverzija***

Prioritetna inverzija se javlja kada je task višeg prioriteta prinuđen da čeka nedefinisani period vremena task nižeg prioriteta da se izvrši. Razmotrimo scenarij sa slijedeće slike: **t1**, **t2** i **t3** su taskovi visokog, srednjeg i niskog prioriteta, respektivno. Task **t3** je pristupio nekom resursu uzimanjem asociranog mu binarnog semafora. Kada **t1** priemptira **t3** i traži resurs uzimanjem istog semafora, on postaje blokiran.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM



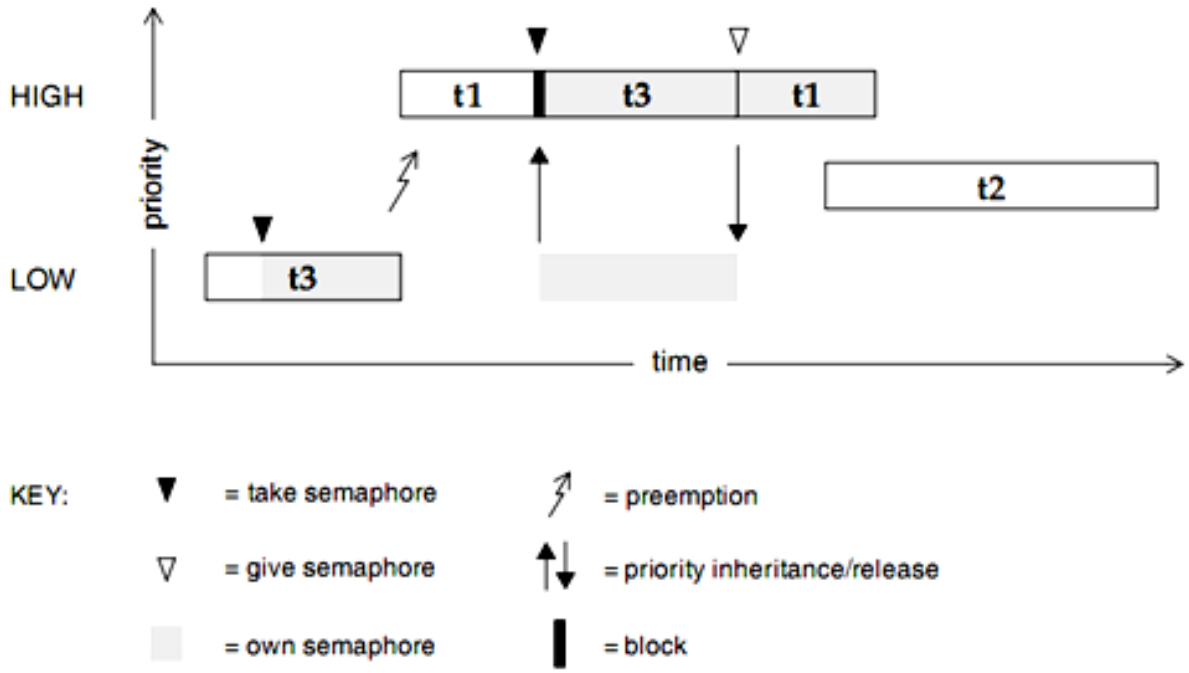

Kad bi smo mogli osigurati da **t1** bude blokiran ne duže od vremena koje je potrebno tasku **t3** da se izvrši, tada ne bi bilo problema jer resurs ne može biti izmješten. Međutim, task niskog prioriteta je osjetljiv na izmještanje od strane taska srednjeg prioriteta **t2**, koji bi mogao spriječiti **t3** da odustane od resursa. Ovako stanje može potrajati, blokiranjem **t1** za nedefinisani period vremena.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

Semafor uzajamnog isključivanja ima opciju **SEM_INVERSION_SAFE** koja uključuje algoritam nasljeđivanja prioriteta. Protokol nasljeđivanja prioriteta osigurava da će se task koji drži resurs izvršavati sa prioritetom taska sa najvišim prioritetom koji je blokiran na tom resursu. Kada je prioritet taska podignut on ostaje na najvišem nivou sve dok se semafori uzajamnog isključivanja koji drže task ne oslobode; tada se task vraća u svoju normalu, ili standardni, prioritet. Stoga, „nasljeđivani“ task je zaštićen od priemptiranja od strane bilo kojeg taska srednjeg prioriteta. Ova opcija mora biti korištena u konjukciji sa redom prioriteta (**SEM_Q_PRIORITY**).

Na slijedećoj slici, nasljeđivanje prioriteta riješava problem prioritetne inverzije podizanjem prioriteta tasku **t3** na prioritet taska **t1** pri čemu je **t1** blokiran na semaforu. Ovo štiti task **t3**, i indirektno taks **t1**, od priemptiranja od strane taska **t2**.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM




Slijedeći primjer kreira semafor uzajamnog isključivanja koji koristi algoritam nasljeđivanja prioriteta:

**semId = semMCreate(SEM_Q_PRIORITY | SEM_INVERSION_SAFE);**

### *Sigurno brisanje*

Još jedan problem kod uzajamnog isključivanja je brisanje taska. Unutar kritičnog regiona zaštičenog sa semaforima, nekad je poželjno da se task zaštiti od neočekivanog brisanja. Brisanje taska koji se izvršava u kritičnom regionu može biti katastrofalno. Resurs bi mogao ostati u zauzetom stanju a semafor koji čuva resurs bi mogao ostati nedostupan, efektivno štiteći sav pristup resursu.

Primitive **taskSafe()** i **taskUnsafe()** daju jedno rješenje za problem brisanja taska. Međutim, semafor uzajamnog isključivanja nudi opciju **SEM_DELETE_SAFE**, koji uključuje implicitni **taskSafe()** sa svakim **semTake()**, i **taskUnsafe()** sa svakim **semGive()**. Na ovaj način, task može biti zaštičen od brisanja za vrijeme dok drži semafor. Ova opcija je efikasnija nego primitive **taskSafe()** i **taskUnsafe()** jer rezultirajući kod zahtijeva nekoliko pristupanja kernelu.

```
semId = semMCreate(SEM_Q_FIFO | SEM_DELETE_SAFE);
```

### *Rekurzivni pristup resursu*

Semafori uzajamnog isključivanja mogu biti uzimani rekurzivno. Ovo znači da semafor može biti uzet više od jednom od strane taska koji ga drži prije nego što konačno bude oslobođen. Rekurzija je korisna za skup rutina koje pozivaju jedna drugu ali koje također zahtijevaju uzajmno isključiv pristup resursu. Ovo je moguće iz razloga što sistem vodi računa o tome koji task trenutno drži semafor.

Prije nego što bude oslobođen, semafor koji je uzet rekurzivno mora biti predat isti broj puta koliko je i uzet. Ovo se prati brojačem koji se povećava sa svakim **semTake()** i umanjuje sa svakim **semGive()**.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

```
  /* Funkcija A zahtijeva pristup resursu što ostvaruje uzimanjem semafora
 * mySem;
  * Funkcija A će također trebati da pozove funkciju B, koja također zahtijeva mySem:
   */
   /* includes */
  #include "vxWorks.h"
  #include "semLib.h"
 SEM_ID mySem;

  /* Kreiraj semafor uzajamnog isključivanja. */
  init ()
{
  mySem = semMCreate (SEM_Q_PRIORITY);
 }
 funcA ()
 {
 semTake (mySem, WAIT_FOREVER);
printf ("funcA: Got mutual-exclusion semaphore\n");
...
```

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

## Brojački semafori

Brojački semafori su još jedan način kako da se implementira sinhronizacija taska i uzajamno isključivanje. Brojački semafor radi kao binarni semafor izuzev što vodi računa o tome koliko je puta semafor dat. Svaki put kada se semafor dadne, brojač se povećava; svaki put kada se semafor uzme, brojač se umanjuje. Kada brojač dođe do nule, task koji pokušava da uzme semafor se blokira. Kao i kod binarnog semafora, ako je semafor dat i task je blokiran, on postaje deblokiran. Međutim, za razliku od binarnog semafora, ako je semafor dat i nema taskova koji su blokirani, tada se brojač povećava. Ovo znači da semafor koji je dat dvaput može biti uzet dvaput bez blokiranja. Naredna tabela prikazuje primjer vremenske sekvence brojanja uzimanja i predaje taskova semafora koji je inicijaliziran na 3.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

| Poziv semafora | Brojač nkon poziva | Rezultirajuče ponašanje |
|---|---|---|
| semCCreate() | 3 | Semafor inicijaliziran sa početnom vrijednošću brojaća od 3 |
| semTake() | 2 | Semafor uzet. |
| semTake() | 1 | Semafor uzet. |
| semTake() | 0 | Semafor uzet. |
| semTake() | 0 | Task blokiran i čeka da semafor postane dostupan. |
| semGive() | 0 | Tasku koji čeka je dat semafor. |
| semGive() | 1 | Nema taska koji čeka na semafor; brojač se povećava za 1. |

Brojački semafori su korisni za zaštitu višestrukih kopija istog resursa. Na primjer, korištenje pet magnetnih drajvova može biti koordinirano korištenjem ove vrste semafora sa inicijalnom vrijednošću brojača od 5, ili prstenastog bafera sa 256 ulaza koji može biti implementiran korištenjem brojačkog semafora sa početnom vrijednošću brojača od 256. Početna vrijednost brojača se predaje kao argument rutini **semCCreate()**.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

## Semafori i VxWorks događaji

U nastavku slijedi opis upotrebe VxWorks događaja sa semaforima. VxWorks događaji se također mogu koristiti i sa drugim VxWorks objektima.

### *Korištenje događaja*

Semafor može tasku poslati događaj, ako se zahtijeva da task učini nešto sa njim. Da bi mogli zahtijevati od semafora da šalje događaje, task se mora prijaviti semaforu korištenjem **semEvStart()**. Tada, svaki put kada je semafor oslobodjen sa **semGive()**, i ako nema drugih taskova na čekanju, semafor šalje događaje registrovanom tasku. Da bi se zaustavilo slanje događaja tasku potrebno je pozvati **semEvStop()**.

Samo jedan task može biti registrovan kod semafora u određenom trenutku. Događaji koje semafor šalje tasku mogu biti vraćeni od strane taska korištenjem rutina u **eventLib**.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

U nekim aplikacijama kreator semafora možda želi da zna kada semafor nije uspio da pošalje događaj. Ovakav scenario se može pojaviti ako se task registruje kod semafora, a zatim se izbriše prije nego što se stigne odjaviti. U ovakvoj situaciji, data operacija može prouzrokovati da semafor pokuša poslati događaje izbrisanom tasku. Takav pokušaj će evidentno pasti. Ako je semafor kreiran sa

**SEM_EVENTSEND_ERROR_NOTIFY** opcijom, data operacija vraća grešku. Inače, VxWorks rukuje greškom neprimjetno.

***Postojeći VxWorks API***

Implementacija VxWorks događaja ne predlaže da se prate svi resursi nad kojima je task trenutno prijavljen. Međutim, resurs može pokušati da šalje događaje tasku koji više ne postoji. Na primjer, task može biti izbrisan ili se može sam uništiti i ostati i dalje prijavljen kod resursa da prima događaje.

Ova greška se javi samo kad resurs postane slobodan. Međutim, u ovom slučaju greška ne znači da semafor nije oslobodjen ili da greška nije pravilno isporučena. To jednostavno znači da resurs nije mogao poslati događaj prijavljenom tasku.

***Uticaj na performanse***

Kad je task na čekanju semafora, nema uticaja na performanse kod **semGive()**. Međutim, ako ovo nije slučaj (na primjer, ako je semafor slobodan), poziv **semGive()** traje duže obzirom da događaji moraju bit poslati tasku. Štaviše, poziv može da ne zavisi od čekanja taska za događajima, što znači da pozivaoc može bit izmješten, čak i ako nema taska koji čeka na semafor. Performanse rutine **semDestroy()** su narušene u slučajevima gdje task čeka događaje od semafora, obzitrom da task treba da bude probuđen. Također treba primjetiti da, u ovom slučaju, događaji ne trebaju biti poslati

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

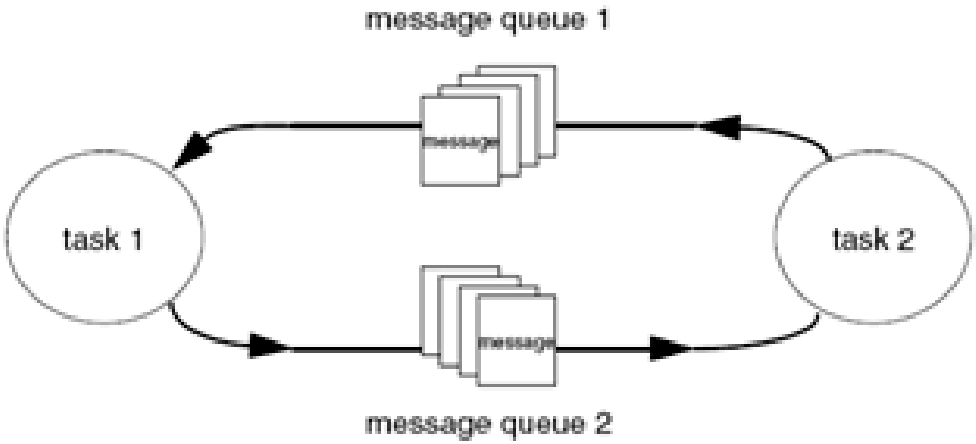
## Redovi poruka

Savremene real-time aplikacije su konstruisane kao skup nezavisnih ali kooperirajučih taskova. Dok semafori pružaju brz mehanizam za sinhronizaciju i zaključavanje taskova, često je mehanizam visokog nivoa neophodan da bi se dozvolilo kooperirajučim taskovima da međusobno komuniciraju. Kod VxWorks operativnog sistema, primarni mehanizam za komunikaciju između taskova unutar jednog procesora su *redovi poruka* ( Message queues).

Redovi poruka dozvoljavaju da varijabilan broj poruka , varijabilne dužine, budu stavljene u red čekanja. Taskovi i ISR-ne mogu poslati poruke u red poruka i taskovi mogu primati poruke iz reda poruka.

Više taskova može poslati i primiti poruku iz istog reda poruka. Full-duplex komunikacija između dva taska obično zahtijeva dva reda poruka, po jedan za svaki smijer – kao na slici

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM




Postoje dvije rutine u VxWorks biblioteci za rad sa redovima poruka. Prva je **msgQLib**, pruža Wind redove poruka dizajnirane isključivo za VxWorks; druga **mqPxLib** koja je kompatibilna sa POSIX standardom.

**Wind redovi poruka**

Wind redovi poruka se kreiraju, koriste i brišu sa rutinama prikazanim u narednoj tabeli. Ova biblioteka daje poruke koje se redaju u FIFO maniru sa jednim izuzetkom: postoje dva nivoa prioriteta i poruke označene sa većim prioritetom

se stavljaju na vrh reda.

| Poziv | Opis |
|---|---|
| msgQCreate() | Alocira i inicijalizira red poruka |
| msgQDelete() | Terminira i oslobađa red poruka |
| msgQSend() | Šalje poruku redu poruka |
| msgQRecieve() | Prima poruku iz reda poruka |

**Pauze**

Rutine **msgQSend()** i **msgQRecieve()** primaju parametre za pauzu – timeout parametre. Kada se šalje poruka, tajm-out određuje koliko će se otkucaja čekati da prostor u baferu postane slobodan, ako nema slobodnog prostora u redu poruka. Kada se primi poruka, tajm-out određuje koliko otkucaja će se čekati kako bi poruka postala dostupna. Kao i sa semaforima, vrijednost tajm-out parametra može imati posebnu vrijednost **NO_WAIT(0)**, što znači da nema čekanja, ili **WAIT_FOREVER(-1)** da rutina nikada ne pauzira.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

## Hitne poruke

Funkcija msgQSend() dozvoljava određivanje prioriteta poruke koji može biti normalni **(MSG_PRI_NORMAL)** ili hitan **(MSG_PRI_URGENT)**. Poruke normalnog prioriteta se stavljaju na kraj liste poredanih poruka, dok se urgentne poruke dodaju na početak liste.

```
/* U ovom jednostavnom primjeru, task t1 kreira red poruka i salje
   poruku tasku t2. Task t2 prima poruku iz reda a zatim je prikazuje.
 */
/* includes */
#include "vxWorks.h"
#include "msgQLib.h"
```

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

```
/* defines */
#define MAX_MSGS (10)
#define MAX_MSG_LEN (100)
MSG_Q_ID myMsgQId;
 task2 (void)
{
 char msgBuf[MAX_MSG_LEN];
 /* dohvati poruku iz reda; ako je potrebno cekaj da poruka postane dostupna*/
 if (msgQReceive(myMsgQId, msgBuf, MAX_MSG_LEN, WAIT_FOREVER) ==
  ERROR)
 return (ERROR);
  /* prikazi poruku */
 printf ("Message from task 1:\n%s\n", msgBuf);
 }
```

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

```
#define MESSAGE "Greetings from Task 1"
task1 (void)
{
/* kreiraj red poruka */
if ((myMsgQId = msgQCreate (MAX_MSGS, MAX_MSG_LEN,
MSG_Q_PRIORITY)) == NULL)
 return (ERROR);
/* posalji poruku normalnog prioriteta, blokiraj ako je red pun */
if (msgQSend (myMsgQId, MESSAGE, sizeof (MESSAGE),
WAIT_FOREVER,
 MSG_PRI_NORMAL) == ERROR)
return (ERROR);
}
```

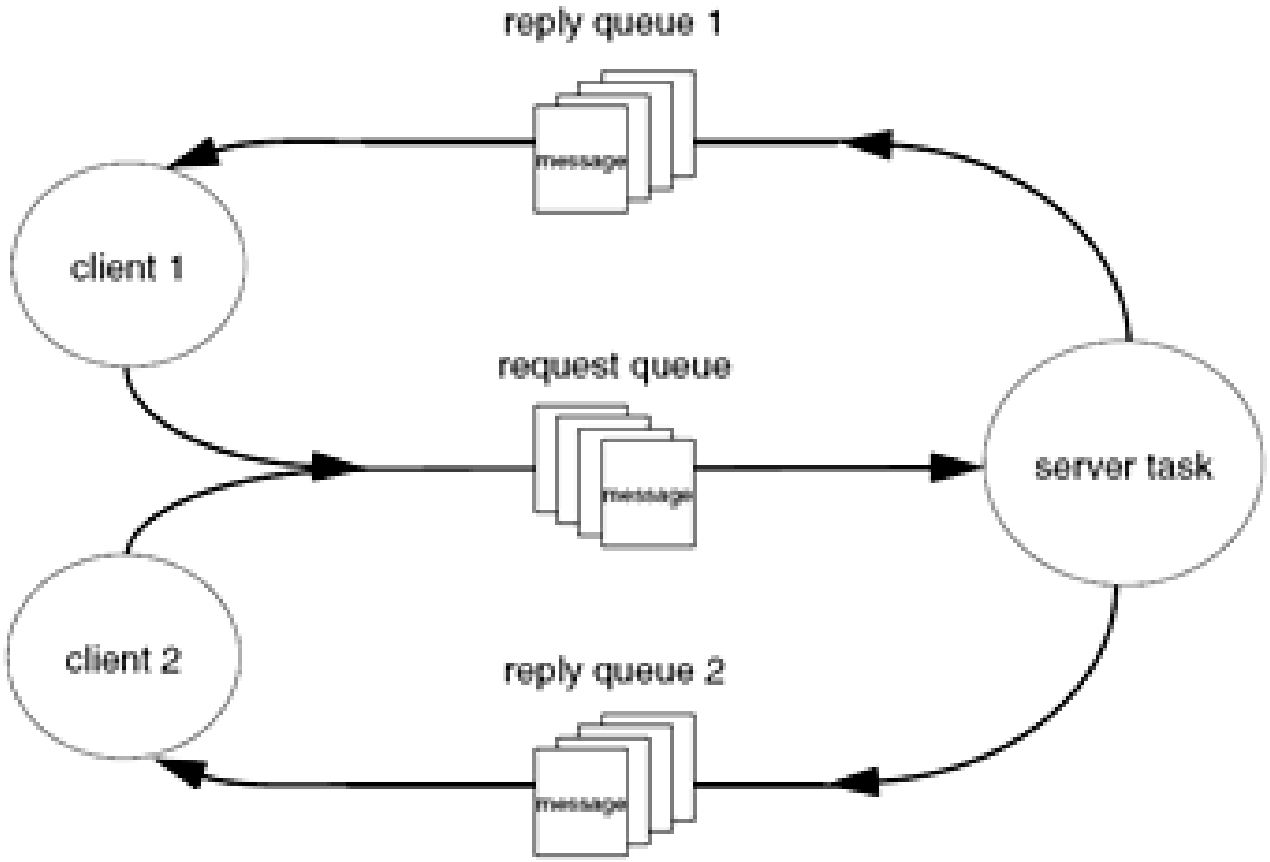
# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

## Serveri i Klijenti sa redovima poruka

Real-time sistemi su obično struktuirani korištenjem server-klijent modela taskova. Kod ovog modela, serverski taskovi prihvataju zahtijeve od klijentskih taskova kako bi obavili određeni posao i obično vratili odgovor. Ovi zahtijevi i odgovori su obično načinjeni u obliku poruka. Kod VxWorksa, redovi poruka ili cijevi su prirodan način da se ovo implementira.

Na primjer, klijent-server komunikacije mogu biti implementirane na način koji je prikazan na narenoj slici. Svaki server task kreira red poruka sa ciljem da primi poruke zahtijeva od klijenta. Svaki klijentski task kreira red poruka kako bi mogao da primi odgovore od servera. Svaki zahtijev sadrži polje **msgQId** od klinetskog odgovora. Glavna petlja serverskog taska se sastoji od čitanja poruka iz reda zahtijevanih poruka, obavljanja zahtijeva i slanja odgovora

klijentskom redu poruka. Ista arhitektura može biti ostvarena sa cijevima umjesto redova poruka.



# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

## *Cijevi*

Cijevi (Pipes) predstavljaju alternativnu redovima poruka koje prolaze kroz VxWorks I/O sistem. Cijevi su virtualni I/O uređaji kojima upravlja **pipeDrv** drajver. Rutina pipeDevCreate() kreira cijev i red poruka koji je pridružen sa tom cijevi. Poziv specificira ime kreirane cijevi, maksimalan broj poruka koje mogu biti naredane i maksimalnu dužinu svake poruke:

**status = pipeDevCreate(„/pipe/name“, max_msgs, max_length);**

Kreirana cijev se obično zove I/O uređaj. Taskovi mogu koristit standardne I/O rutine da otvore, čitaju i upisuju u cijevi, i pozivaju *ioctl* rutine. Kao što rade sa ostalim I/O uređajima, taskovi se blokiraju kada čitaju iz prazne cijevi sve dok podaci ne postanu dostupni, i blokiraju se kada upisuju u punu cijev sve dok se prostor ne oslobodi. Kao i kod redova poruka, servisne rutine prekida mogu upisivati u cijev ali ne mogu čitati

iz cijevi.

Kao I/O uređaji, cijevi pružaju bitno svojstvo koje redovi poruka nemaju – mogućnost korištenja sa **select()** rutinom. Ova rutina dozvoljava tasku da čeka podatke da budu dostupni na bilo kojem skupu I/O uređaja. Rutina **select()** također radi sa drugim asinhronim I/O uređajima uključujući mrežne sokite i serijske uređaje. Dakle, korištenjem **select()** rutine, task može čekati podatak na kombinaciji nekoliko cijevi, sokita i serijskih uređaja. Također, cijevi nam dozvoljavaju da implementiramo klijent-server model za komunikaciju između taskova.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

## *Mrežna komunikacije između taskova*

Kod VxWorks real-time operativnog sistema osnova komunikacije između taskova se odvija preko mrežnih sokita. Sokit je krajnja tačka u komunikaciji između taskova; podaci se šalju iz jednog sokita u drugi. Kada kreiramo sokit, specificiramo protokol Internet komunikacije koji prenosi podatke. VxWorks podržava Internet protokole TCP i UDP.

Jedna od najvećih prednosti komunikacije sokitima je da je „homogena“. Komunikacija sokitima između procesa je identična bez obzira na lokaciju procesa u mreži ili operativnog sistema nad kojim se izvršava. Procesi mogu komunicirati u sklopu jednog procesora, preko Etherneta ili preko bilo koje vrste mrežne komunikacije. Komunikacija preko sokita se može javiti između VxWorks taskova i host sistemskih procesa u bilo kojoj kombinaciji. U svim slučajevima, komunikacija izgleda identično za aplikaciju,

izuzev, naravno, brzine.

**Daljinski poziv procedure(Remote Procedure Calls-RPC)**

Daljinski poziv procedure (RPC) je mogućnost koja dozvoljava procesu na jednoj mašini da pozove proceduru koja je pokrenuta od strane drugog procesa na istoj ili udaljenoj mašini. Interno, RPC koristi sokite kao mehanizam komunikacije. Dakle, sa RPC-om, VxWorks taskovi i procesi na host sistemu mogu pozvati rutine koje će se izvršiti na drugom VxWorks sistemu ili host mašini, u bilo kojoj kombinaciji.

***Signali***

VxWorks podržava softverske signale. Signali asinhrono mijenjaju kontrolni tok taska. Bilo koji task ili ISR može generisati signal za određeni task. Task prima signal i odmah suspenduje tekuću nit koja se izvršava i pokreće specifičnu

rutinu za rukovanje signalom. Rutina za rukovanje signalom se izvršava u kontekstu taska koji prima signal. Rukovatelj signalom se poziva čak i kada je task blokiran.

Wind kernel podržava dvije vrste signala: UNIX BSD signale i POSIX kompatibilne signale. POSIX kompatibilni signalni interfejs, zauzvrat, sadrži i osnovni signalni interfjes specificiran u POSIX standardu 1003.1, i standardne signale iz POSIX 1003.1b.

**Osnovne signalne rutine**

VxWorks koristi osnovnu signalnu komponentu **INCLUDE_SIGNALS**. Ova komponenta automatski inicijalizira signale sa **sigInit()**. Naredna tabela prikazuje osnovne signalne rutine.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

| POSIX 1003.1b kompatibilni poziv | UNIX BSD kompatibilni poziv | Opis |
|---|---|---|
| signal() | signal() | Specificira rutinu za rukovanje pridruženu sa signalom. |
| kill() | kill() | Šalje signal tasku. |
| raise() | N/A | Šalje signal samom sebi. |
| sigaction() | sigvec() | Istražuje ili postavlja rutinu za rukovanje signalom za određeni signal. |
| sigsuspend() | pause() | Suspenduje task dok se signal ne isporući. |
| sigpending() | N/A | Prima skup blokiranih signala |
| sigemptyset() | sigsetmask() | Manipulira maskom signala. |
| sigfillset() | | |
| sigaddset() | | |
| sigdelset() | | |
| sigismember() | | |
| sigprocmask() | sigsetmask() | Postavlja masku blokiranih signala. |
| sigprocmask() | sigblock() | Dodaje skupu blokiranih signala. |

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

Na mnoge načine, signali su analogni hardverskim interaptima. Postoji 31 različit signal. Rukovatelj signalom se veže za određeni signal sa **sigvec()** ili **sigaction()**. Signal može biti poslat tasku pozivanjem rutine **kill()**. Ovo je analogno pojavljivanju interapta. Rutine **sigsetmask()** i **sigblock()** ili **sigprocmask()** puštaju da signali budu selektivno onemogućeni.

Određeni signali su povezani sa hardverskim izuzecima. Na primjer, greške sabirnice, ilegalne operacije i izuzetci vezani za pokretne zareze uzrokuju specifične signale.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

## Upravljanje memorijom

Predstavlja upravljanje memorijom dostupno aplikacijama koje se izvršavaju kao procesi u realnom vremenu. Svaki proces ima svoj vlastiti heap, i može alocirati i osloboditi spremnike od svog heap-a sa rutinama obezbjeđenim u memLib i memPartLib bibliotekama. Osim toga, objekti za run-time otkrivanje grešaka obezbjeđuju sposobnost za debugiranje grešaka vezanih za memoriju u kodu aplikacije. Korisničke aplikacije također mogu koristiti sljedeće objekte za djeljenu memoriju i mapiranje memorije:

- Lokalne VxWorks djeljene regije podataka
- POSIX objekti za upravljanje memorijom

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

## Konfiguracija VxWorksa sa upravljanjem memorije

Kada je VxWorks konfigurisan sa podrškom procesa u realnom vremenu, on uključuje osnovne izvedbe upravljanja memorijom potrebne za procesno bazirane aplikacije. Za karakteristike otkrivanja grešaka obezbjeđene sa heap-om i opremom za particionisanje memorije, potrebna je takodjer i komponenta INCLUDE_PER_RTP_SHOW.

## VxWorks RTP modeli virtualne memorije

VxWorks može biti podešen da koristi ili ravni (flat) ili preklapani (overlapped) model virtualne memorije za RTP. Po defaultu, on koristi ravni model virtualne memorije.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

## Ravni RTP virtualni memorijski model

Ravni virtualni memorijski model je defaultni i ne zahtijeva neku određenu konfiguraciju. Sa ovim modelom svaki VxWorks proces ima svoj dio virtualne memorije - procesi se ne preklapaju u virtualnoj memoriji. Ravna mapa virtualne memorije obezbjeđuje sljedeće prednosti:

- Brzina – prebacivanje konteksta je brzo
- Lakoća debugiranja – adrese za svaki proces su jedinstvene.
- Fleksibilan model programiranja koji obezbjeđuje isti procesni model bez obzira na podršku MMU. VxWorks aplikacioni memorijski model omogućava pokretanje istih aplikacija sa i bez MMU.

Sistemi mogu biti razvijeni i debugirani na odredištima sa MMU, i zatim biti dostavljeni na hardware koji ne posjeduje MMU.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

Prednost kod ovakvog načina razvoja aplikacije je olakšano debugiranje u razvoju, manji troškovi isporučenih jedinica, kao i prednosti performansi ciljnih konfiguracija bez MMU, ili u odnosu na slučaj kada MMU nije omogućena. Sa ravnim virutalnim memorijskim modelom, ipak, izvršni fajlovi moraju biti premješteni, što znači da je vrijeme njihovog punjenja sporije nego sa preklapajućim modelom. Pored toga, ravni model ne dozvoljava selekciju određenih dijelova virtualne memorije u kojim su instalirani tekst, podaci, bss djelovi aplikacije realnog vremena. Umjesto najboljeg odgovarajućeg područja, algoritam automatski odabire područje virtualne memorije koje najviše odgovara, bazirano na trenutnoj upotrebi memorije, gdje ti segmenti trebaju biti napunjeni.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

**Virtualni memorijski model RTP-a sa preklapanjem**

VxWorks može biti konfigurisan da koristi preklapani virtualni model umjesto ravnog modela. Sa preklapnim modelom sve VxWorks aplikacije djele zajednički opseg virtualnog adresnog prostora u kojem su instalirani tekst, podaci, i bss segmenti, i sve aplikacije djele izvršne adrese.

Pored toga, da bi RTP aplikacije zadržavale tekst, podatke, i bss segmente, preklapani dio virtualne memorije je automatski korišten za bilo koji element koji je privatan procesu, kao što je heap i skup taskova (sve dok postoji dovoljno prostora, u suprotnom će biti van prostora).

- Da bi iskoristili prednosti preklapanog modela, RTP aplikacije moraju biti pravljene kao apsolutno povezani izvršni fajlovi. One djele iste izvršne adrese, koje su unutar preklapanog područja virtulane memorije. Zbog jednostavnosti, izvršne

izvršne adrese bi trebale biti definisane u bazi ovog područja. Kada su napunjene, korak premještanja je nepotreban.

Primjetimo da ako je nekoliko instanci iste RTP aplikacije pokrenuto u sistemu, one zauzimaju tačno isti opseg virtualnih memorijskih adresa, ali svaka instanca i dalje ima fizičku memoriju alociranu za svoj tekst, podatke, i bss segmente. Drugim riječima, ne postoji dijeljenje tekstualnog segmenta između više instanci aplikacije.

Preklapani virtualni model obezbjeđuje sljedeće prednosti:

- Brže vrijeme punjenja u memoriju kada su aplikacije napravljene kao apsolutno-povezane i izvršne. Faza premještanja je preskočena kada su aplikacije apsolutno-povezane i izvršne, zato što su sve instalirane na iste izvršne adrese virtualne memorije. Poboljšanje u vremenu punjenja je reda 10% - 50%. Poboljšanja u vremenu punjenja su ovisna o

broju premještenih unosa u fajl.

- Preciznija kontrola sistemske virtualne memorije. Korisnik odabire područje u virtualnoj memoriji u kojem su instalirani tekst, podaci i bss segmenti RTP aplikacija. Sa ravnim memorijskim modelom, s druge strane, algoritam najboljeg odgovaranja automatski odabire najprikladnije područje slobodne virtualne memorije, bazirano na trenutnoj memorijskoj iskorištenosti, gdje će ti segmenti biti napunjeni.
- Učinkovitija upotreba sistema virtualne memorije smanjuje mogućnost fragmentacije memorije.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

## I/O Sistem

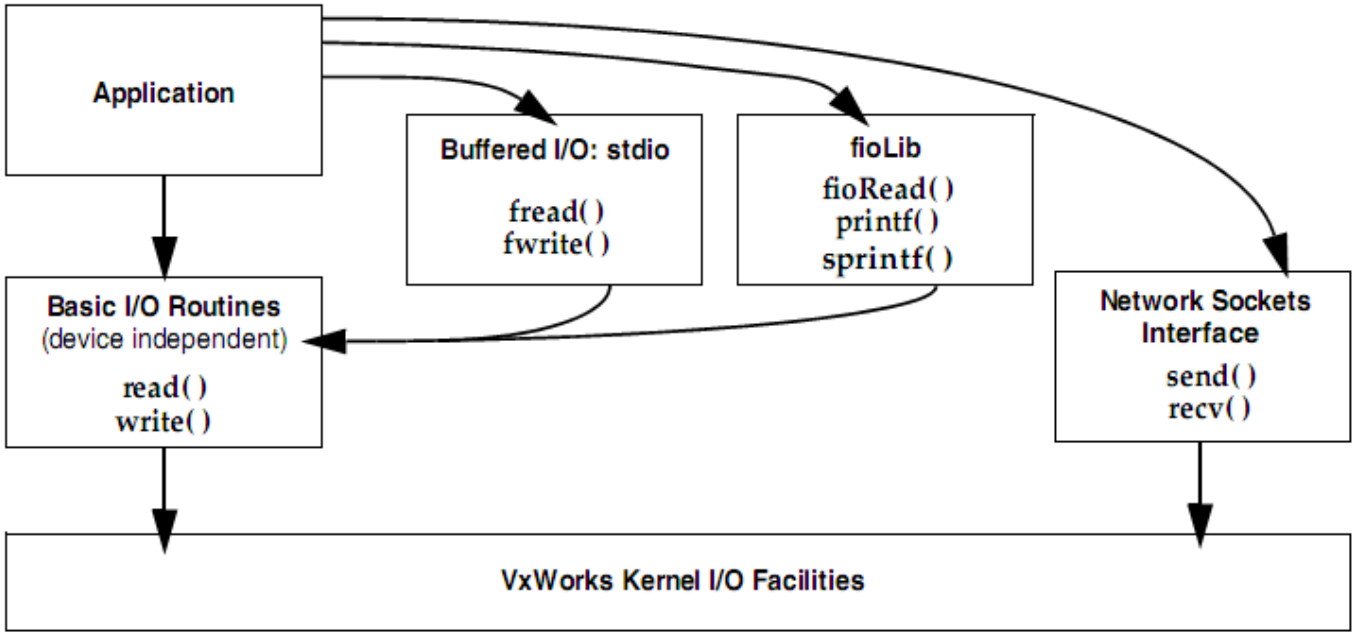
VxWorks I/O sistem je napravljen da predstavi jednostavan, uniforman, uređajno neovisan interfejs o bilo kojem uređaju, uključujući:

Karakter orijentisane uređaje kao što su terminali ili komunikacijske linije.

- Slučajni pristup (random-access) blokira uređaje kao što su diskovi.
- Virtualni uređaji kao što su intertask cijevi i adapteri.
- Nadgledanje i kontrola uređaja kao što su digitalni i analogni I/O uređaji.
- Mrežni uređaji koji daju pristup udaljenim uređajima.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

VxWorks I/O sistem obezbjeđuje standardne C bibiloteke i za osnovne i za bufferovane I/O. Osnovne I/O biblioteke su UNIX kompatibilne, Buferovane I/O biblioteke su ANSCI C-komaptibilne. Dijagram na narednoj slici prikazuje odnos između različitih elemenata VxWorks I/O sistema dostupnih procesima realnog vremena (RTP).



# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

## Lokalni fajl sistem

VxWorks obezbjeđuje raznolikost fajl sistema koji su prikladni za različite tipove aplikacija. Fajl sistemi mogu biti korišteni istovremeno, i u većini slučajeva u više instanci, za jedan VxWorks sistem.

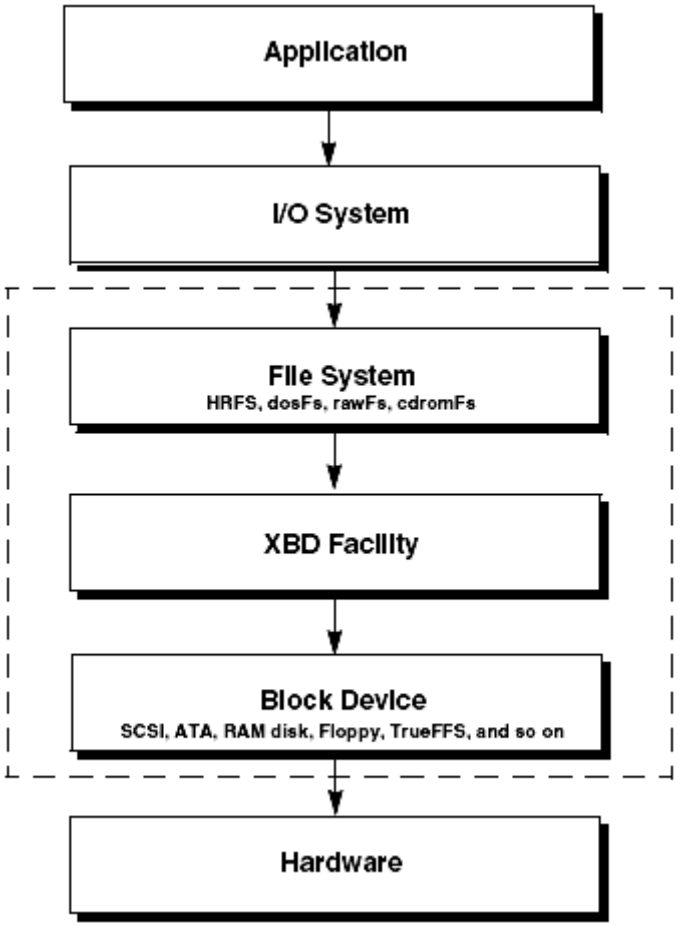
Većina VxWorks fajl sistema se oslanja na proširene objekte bloka uređaja (XDB) za standardni I/O interfejs između fajl sistema i drivera uređaja. Ovaj standardni interfejs dozvoljava pisanje vlastitog sistema za VxWorks, i slobodno miješa fajl sistem i drivere uređaja.

Fajl sistem korišten za prenosive uređaje koristi monitor fajl sistema za automatsku detekciju umetanja uređaja i instanciranje odgovarajućeg fajl sistema na uređaju.

Relacija između aplikacija, fajl sistema, I/O objekata, drivera uređaja i hardverskih uređaja je prikazana na narednoj slici

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

Ova struktura je relevantna za HRFS, dosFs, rawFs, i cdromFs fajl sisteme. Isprekidana linija označava elemente koji moraju biti konfigurisani i instancirani da kreiraju specifičan, funkcionalan run-time fajl sistem.



# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

U nastavku ćemo objasniti sljededeće VxWorks fajl sisteme i kako su oni korišteni:

- VRFS, Virtualni Root Fajl Sistem za upotrebu sa aplikacijama koje koriste POSIX root fajl sistem. VRFS je jednostavno root direktorij iz kojeg drugim fajl sistemima i uređajima može biti pristupano.
- HRFS, POSIX komplementaran transakcioni fajl sistem dizajniran za real-time upotrebu bloka uređaja (diskova). Može biti korišten na flash memoriji zajedno sa TrueFFS i XDB blok wrapper komponentom.
- dosFs, MS-DOS komaptibilan fajl sistem dizajniran za real time upotrebu bloka uređaja. Može bit korišten na flash memorijama zajedno sa TrueFFS i XDB blok wrapper komponentom. Također može biti korišten sa transakciono baziranim pouzdanim fajl sistemom (TRFS

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

- rawFs, obezbjeđuje jednostavni raw fajl sistem koji tretira disk kao jedan veliki fajl.
- cdromFs, omogućava aplikacijama da čitaju podatke sa CD-ROM-ova formatiranih prema ISO 9660 standardu fajl sistema.
- ROMFS, dizajniran za povezivanje aplikacija i drugih fajlova sa image-om VxWorks sistema. Medij za pohranjivanje nije potreban, osim onog koji se koristi za podizanje VxWorks image-a.
- TSFS, koristi host odredišni server da bi obezbjedio odredište sa pristupom fajlovima na host sistemu.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

## VxWorks događaji

VxWorks događaji su način komunikacije između taskova i interaptnih rutina (ISR), između taskova, ili između taskova i VxWorks objekata. U kontekstu VxWorks događaja, ovi objekti se nazivaju resursima, i oni sadrže semafore i redove poruka. Samo taskovi mogu primiti događaje; dok taskovi, interaptne rutine ili resursi ih mogu i poslati.

Sa ciljem da primi događaj iz resursa, task se mora prijaviti kod resursa. Da bi resurs mogao slati događaje, resurs mora biti slobodan. Komunikacija između taskova i resursa je na peer-to-peer osnovi, što znači da samo registrovani task može primiti događaje iz resursa. U tom smislu, događaji su kao signali, stim što su usmjereni na samo jedan task. Task može čekati na događaje iz više izvora; prema tome, može čekati na semafor da postane slobodan i na poruku da stigne u red poruka

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

Događaji su po prirodi sinhroni (za razliku od signala), što znači da task koji prima događaj mora biti blokiran ili na čekanju dok se ne desi događaj. Kada se primi željeni događaj, task koji čeka mora nastaviti sa izvršavanjem, kao što bi nakon poziva **msgQRecieve()** ili **semTake()**. Stoga, za razliku od signala, događaji ne zahtijevaju rutinu za rukovanje istim.

Taskovi također mogu čekati na događaje koji nisu povezani sa resursima. To su događaji koji su poslati iz drugog taska ili interaptne rutine. Task se ne registruje da bi primio ove događaje; task pošiljaoc ili ISR jednostavno moraju znati zašto task želi da prima događaje.

Značenje svakog događaja se razlikuje za svaki task. Na primjer, kada se primi eventX, on može biti interpretiran različito od strane svakog taska koji ga primi. Također, jednom kad je događaj primljen od strane taska, događaj će biti

ignorisan ako bude poslat istom tasku. Zbog toga nije moguće pratiti koliko je puta svaki događaj bio poslat tasku.

***pSOS Događaji***

U nastavku slijedi opis funkcionalnosti pSOS događaja. Ova funkcionalnost omogućava osnove VxWorks događaja, ali ne opisuje u potpunosti njihovo ponašanje.

**Slanje i primanje događaja**

Kod pSOS operativnog sistema događaji mogu biti poslani iz resursa do taska, iz interaptne servisne rutine (ISR) do taska ili direktno između dva taska. Taskovi, interptne rutine i resursi svi koriste isti **ev_send()** API za slanje događaja.

Da bi task primio događaj iz resursa, task se mora registrovati kod tog resursa i zahtijevati da mu pošalje poseban skup događaja kada resurs postane slobodan. Resurs je ili semafor ili red poruka. Kada resurs postane slobodan, on šalje skup

događaja registrovanom tasku. Ovaj task može, ali i ne mora, čekati na događaje.

Kao što je spomenuto ranije, task također može primiti događaje od drugog taska. Na primjer, ako se dva taska slože da šalju događaje između sebe, taskA može poslati tasku taskB specifični skup događaja kada (taskA) završi sa izvršavanjem, kako bi taskB znao da se to dogodilo. Kao i kod događaja poslatih iz resursa, task koji prima može a i ne mora čekati na događaje.

**Čekanje na događaje**

Task može čekati na nekoliko događaja iz jednog ili više izvora. Svaki izvor može poslati više događaja, i task također čeka da primi samo jedan događaj, ili sve događaje. Na primjer, task može čekati na događaje 1 do 10, gdje 1 do 4 dolaze od semafora, 5 do 6 dolaze od reda poruka, i 7 do 10

dolaze od drugog taska.

**Prijavljivanje za događaje**

Samo jedan task može samog sebe registrovati da prima događaje iz resursa. Ako se zatim drugi task registruje sa istim resursom, prethodno registrovani task se automatski odjavljuje, bez njegovog znanja. Ovakvo ponašanje se razlikuje od VxWorks događaja.

Kada se task prijavi kod resursa, događaji se ne šalju odmah, čak i ako je resurs slobodan u trenutku prijavljivanja. Događaji se šalju tek slijedeći put kad resurs postane slobodan.

**Oslobađanje resursa**

Kad resurs pošalje događaj tasku da naznači da je slobodan, to ne znači da je resurs rezervisan. Međutim, čekanje taska na događaje iz resursa će biti prekinuto kada resurs postane slobodan, iako resurs može biti ponovo zauzet u

međuvremenu. Ne postoji garancija ta će resurs uvijek biti dostupan, iako će task kasnije pokušavati da zauzme resurs.

## API pSOS Događaja

API rutine pSOS događaja su prikazane u narednoj tabeli:

| Rutina | Značenje |
|---|---|
| ev_send() | Šalje događaj tasku. |
| ev_recieve() | Čeka događaje. |
| sm_notify() | Prijavljuje task da bude obaviješten o dostupnosti semafora. |
| q_notify() | Prijavljuje task da bude obaviješten o stizanju poruke u red poruka. |
| q_vnotify() | Prijavljuje task da bude obaviješten o dolasku poruke na red poruka varijabline dužine. |

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

## *VxWorks događaji*

Implementacija VxWorks događaja je zasnovana na načinu na koji rade pSOS događaji. U nastavku će biti riječi o nekim bitnim terminima koji se koriste kada su u pitanju VxWorks događaji.

### Definicija slobodnog resursa

Ključni koncept kod razumijevanja događaja poslatih od strane resursa, je da resurs šalje događaje kad postane slobodan. Dakle, bitno je definisati šta to znači da resurs bude *slobodan* kod VxWorks događaja.

## *Mutex semafor*

Mutex semafor se smatra slobodnim kada nema vlasnika i niko ne čeka na njega. Na primjer, nakon poziva **semGive()**, semafor neće poslati događaje ako neki task čeka sa semTake() za isti semafor.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

***Binarni semafor***

Binarni semafor se smatra slobodnim kada nema taskova koji ga drže i niko ne čeka na njega.

***Brojački semafor***

Brojački semafor se smatra slobodnim kada je vrijednost njegovog brojača različita od nule i nema niko ko čeka na njega. Dakle, događaji se ne mogu koristiti kao mehanizam za izračun koliko je puta semafor oslobođen ili dat.

***Red poruka***

Red poruka se smatra slobodnim kada ima poruka u redu i neko ne čeka na dolazak poruke u taj red. Dakle, događaji ne mogu biti korišteni kao mehanizam da se izračuna broj poslatih poruka u red poruka.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

## VxWorks poboljšanja u odnosu na pSOS događaje

Kada su implementirani VxWorks događaji, napravljena su poboljšanja u odnosu na osnovnu pSOS funkcionalnost

- **Prijava jednog taska resursu.** Kada se task prijavi resursu da mu šalje pSOS događaje, on može odjaviti drugi task koji se prethodno prijavio kod resursa. Ovo štiti prvi task od primanja događaja od resursa nad kojim se registrovao. Posljedica toga je da task koji se prvi prijavio može ostati u beskonačnom stanju čekanja. Sa ciljem da se riješi ovaj problem, VxWorks događaji pružaju jednu opciju kojom drugom tasku nije dopušteno da se prijavi resursu ako je neki drugi task već prijavljen kod njega. Ako drugi task pokuša da se prijavi, vraća se greška.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

- **Opcija za trenutno Slanje.** Kada se pSOS task prijavi kod resursa, resurs ne treba odmah da šalje događaje tasku, čak i ako je slobodan u trenutku prijave. Za VxWorks događaje, osnovno ponašanje je isto. Međutim, VxWorks događaji pružaju opciju koja dozvoljava tasku, da u vrijeme prijave, zahtijeva od resursa hitno slanje događaja, ako je resurs slobodan u vrijeme prijave.
- **Opcija za automatsko odjavljivanje.** Postoje situacije u kojima će task imati potrebu da primi događaje od resursa samo jedanput, da bi se zatim odjavio. pSOS implementacija zahtijeva od taska da se eksplicitno odjavi nakon što primi događaj iz resursa. VxWorks implementacija daje opciju da task koji se prijavljuje može reči resursu da mu pošalje događaj samo jedanput i da ga zatim automatski odjavi.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

- **Automatsko sklanjanje pri brisanju resursa.** Kada je resurs (semafor ili red poruka) izbrisan, **semDelete()** i **msgDelete()** odjavljuju bilo koji task. Ovo štiti task od beskonačnog čekanja dok čeka na događaje iz resursa koji je već izbrisan. Čekajući task nastavlja izvršavanje i prima **ERROR** vrijednost iz **eventRecieve()** koji je uzrokuje da task čeka.

**Registar događaja taska**

Svaki task ima svoje vlastito polje ili kontejner, poznat kao registar događaja taska (Task events register). Ovaj registar je 32-bitno polje koje se koristi za čuvanje događaja koje task prima od resursa, interpatne servisne rutine ili drugih taskova. Ovome registru se ne pristupa direktno. Taskovi, interaptne rutine i resursi pune ovaj registar tako što šalju događaje tasku. Task također može sebi da šalje događaje puneći na taj način vlastiti registar

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

Događaji 25 do 32 (VXEV25 ili 0x01000000 do VXEV32 ili 0x80000000) su rezervisani za sistem i nisu dostupni za VxWorks korisnike. Tabela 16 opisuje rutine koje utiču na sadržaj ovog registra.

| Rutina | Efekti |
|---|---|
| eventRecieve() | Čisti ili ostavlja sadržaj registra događaja netaknutim u zavisnosti od odabrane opcije. |
| eventClear() | Čisti sadržaj registra događaja. |
| eventSend() | Kopira događaj u registar događaja. |
| semGive() | Kopira događaje u registar događaja, ako je task prijavljen kod semafora. |
| msgQSend() | Kopira događaje u registar događaja, ako je task prijavljen kod reda događaja. |

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

## API VxWorks događaja

Za svrhu debagiranja sistema koji koristi događaje, taskShow, semShow i msgShow biblioteke prikazuju informacije o događajima.

Biblioteka **taskShow** prikazuje slijedeće informacije:

- Sadržaj registra događaja
- Željene događaje
- Specificirane opcije kada se pozove eventRecieve()

Biblioteke **semShow()** i **msgQShow()** prikazuju slijedeće informacije:

- Task koji je prijavljen da prima događaje
- Izvor događaja koji je označen za slanje događaja tasku
- Opcije predate **semEvStart()** ili **msgQEvStart()** rutinama

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

***Watchdog tajmeri***

VxWorks sadrži mehanizam watchdog tajmera koji dopušta bilo kojoj C funkciji da bude spojena određeni period vremena. Watchdog tajmeri se održavaju kao dio interpatne rutine sistemskog sata.

Funkcije pozvane od strane watchdog tajmera se izvršavaju kao interaptni kod na nivou interapata sistemskog sata. Međutim, ako je kernel u nemogućnosti da trenutno izvrši funkciju iz određenog razloga (kao što je prethodni interapt ili stanje kernela), funkcija se stavlja u **tExcTask** radni red. Funkcije u **tExcTask** redu se izvršavaju prema nivou prioriteta (obično 0).

Funkcije u slijedećoj tabeli su date od strane **wdLib** biblioteke.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

| Poziv | Opis |
|---|---|
| wdCreate() | Alocira i inicijalizira watchdog tajmer. |
| wdDelete() | Uklanja i dealocira watchdog tajmer |
| wdStart() | Pokreće watchdog tajmer. |
| wdCancel() | Otkazuje trenutni watchdog tajmer. |

Watchdog tajmer se prvo kreira pozivanjem **wdCreate()** rutine. Zatim tajmer može biti pokrenut pozivanjem **wdStart()** rutine koja kao argumente uzima broj otkucaja do isteka, C funkciju koju treba pozvati i argument koji se prosljeđuje toj funkciji. Nakon što istekne određeni broj otkucaja, poziva se funkcija sa datim argumentom. Watchdog tajmer može biti otkazan u bilo koje vrijeme prije nego što istekne pozivom rutine **wdCancel()**.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

```
 /* Kreira watchdog tajmer I setuje kad a istekne za 3 sekunde.*/
/* includes */
 #include "vxWorks.h"
 #include "logLib.h"
 #include "wdLib.h"
  /* defines */
  #define SECONDS (3)
 WDOG_ID myWatchDogId;
   task (void)
   {
   /* Kreiraj watchdog */
  if ((myWatchDogId = wdCreate( )) == NULL)
   return (ERROR);
  /* Podesi tajmer da istekne za  SECONDS – ispisujuci poruku na stdout */
  if (wdStart (myWatchDogId, sysClkRateGet( ) * SECONDS, logMsg,
   "Watchdog timer just expired\n") == ERROR)
  return (ERROR);
/* ... */
  }
```

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

***Interrupt Service Code: ISR***

Rukovanje hardverskim izuzecima je ključno kod real-time sistema iz razloga što je obično preko izuzetaka sistem informisan o vanjskim događajima. Za najbrži mogući odgovor na izuzetke, VxWorks pokreće servisne rutine prekida (ISR) u posebnom kontekstu koji je izvan konteksta bilo kojeg taska. Dakle, rukovanje izuzecima ne uključuje izmjenu konteksta. Tabela prikazuje spisak rutina datih u **intLib** i **intArchLib**.

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

| Poziv | Opis |
|---|---|
| intConnect() | Spaja C rutinu sa vektorom interapta. |
| intContext() | Vraća TRUE ako se poziva se nivoa interapta. |
| intCount() | Vraća trenutnu dubinu ugnježđavanja interapta. |
| intLevelSet() | Postavlja nivo maske interapta procesora. |
| intLock() | Onemogućuje interapte. |
| intUnlock() | Ponovo omogućuje interapte. |
| intVecBaseSet() | Postavlja baznu adresu vektora. |
| intVecBaseGet() | Vraća baznu adresu vektora. |
| intVecSet() | Postavlja vektor izuzetaka. |
| intVecGet() | Vraća vektor izuzetaka. |

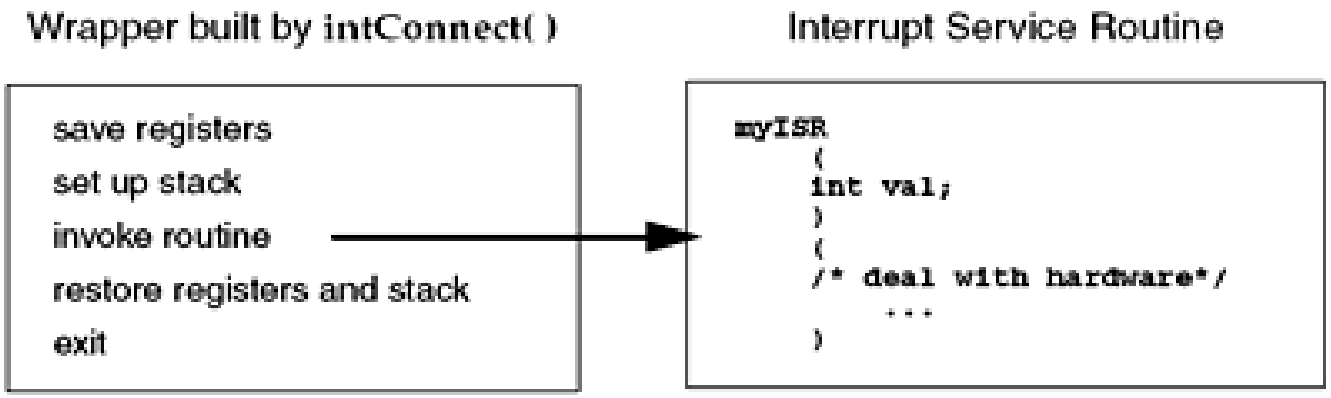
## *Spajanje rutina sa prekidima*

Možemo koristiti i druge hardverske prekide osim onih koje se koriste od strane VxWorksa. VxWorks pruža rutinu **intConnect()** koja dozvoljava C funkcijama da budu spojene na bilo koji prekid. Argumenti za ovu rutinu su byte offset vektora prekida na koji se spaja, adresa C funkcije koja će se spojiti i argument koji se prosljeđuje funkciji. Kada se dogodi prekid sa vektorom uspostavljenim na ovaj način, spojena C

# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

funkcija se poziva na nivou prekida sa specificiranim argumentom. Kada se rukovanje prekidom završi, vraćamo se iz spojene funkcije. Rutina spojena na prekid na ovaj način zove se *interrupt service routine* (ISR).

Prekidi ne mogu povezati vektor direktno sa C funkcijama. Umjesto toga, **intConnect()** pravi mali dio koda koji snima neophodne registre, postavlja stek sa proslijeđenim argumentima i poziva spojenu funkciju. Pri povratku iz funkcije on vraća stanja registara i izlazi iz prekida – slijedeća slika



# RAZVOJ APLIKACIJA ZA VXWORKS OPERATIVNI SISTEM

## *Komunikacija interrupt-task*

Dok je bitno da VxWorks podržava direktnu konekciju ka interrupt service rutinama koje se izvršavaju na nivou prekida, događaji prekida se obično propagiraju do nivoa koda taska. Mnoge VxWorks mogućnosti nisu dostupne kodu na nivou prekida, uključujući I/O ka bilo kojem uređaju osim cijevi. Slijedeće tehnike mogu biti korištene za komunikaciju iz interrupt service rutina do koda na nivou taska:

- Dijeljena memorija i prstenasti baferi.
- Semafori.
- Redovi poruka.
- Cijevi.
- Signali.