Programare concurentă în C (IV) :

Gestiunea proceselor, partea a II-a: Reacoperirea proceselor – primitivele `exec( )`

Cristian Vidraşcu

vidrascu@info.uaic.ro

Sumar

- Introducere
- Primitivele din familia `exec`
- Caracteristicile procesului după `exec`
- Exemplu (1): Reacoperirea procesului
- Exemplu (2): Reacoperire recursivă
- Exemplu (3): Reacoperire cu fişiere deschise
- Exemplu (4): Redirectarea `stdout`-ului

Introducere

Singura modalitate de a crea un nou proces în `UNIX` este prin apelul funcţiei `fork`. Numai că în acest fel se creează o copie a procesului apelant, adică o nouă instanţă de execuţie a aceluiaşi fişier executabil.

Şi atunci, cum este posibil să executăm un alt fişier executabil decât cel care apelează primitiva `fork`?

Introducere

Singura modalitate de a crea un nou proces în `UNIX` este prin apelul funcţiei `fork`. Numai că în acest fel se creează o copie a procesului apelant, adică o nouă instanţă de execuţie a aceluiaşi fişier executabil.

Şi atunci, cum este posibil să executăm un alt fişier executabil decât cel care apelează primitiva `fork`?

Răspuns: prin utilizarea unui alt mecanism, acela de "*reacoperire a proceselor*", disponibil în `UNIX` prin intermediul primitivelor de tipul `exec`.

Primitivele din familia exec

Familia de primitive exec transformă procesul apelant într-un alt proces specificat (prin numele fişierului executabil asociat) ca argument al apelului exec.

Noul proces se spune că "*reacoperă*" procesul ce a executat apelul exec, şi el moşteneşte caracteristicile acestuia (inclusiv PID-ul), cu excepţia a câtorva dintre ele.

Primitivele din familia exec

Familia de primitive exec transformă procesul apelant într-un alt proces specificat (prin numele fişierului executabil asociat) ca argument al apelului exec.
Noul proces se spune că "*reacoperă*" procesul ce a executat apelul exec, şi el moşteneşte caracteristicile acestuia (inclusiv PID-ul), cu excepţia a câtorva dintre ele.

Există în total 6 funcţii din familia exec. Ele diferă prin nume şi prin lista parametrilor de apel, şi sunt împărţite în 2 categorii, ce se diferenţiază prin forma în care se dau parametrii de apel:

- numărul de parametri este variabil
- numărul de parametri este fix

Primitivele din familia `exec` (cont.)

1) Prima pereche de primitive `exec` este perechea `execl` şi `execv`, cu interfeţele următoare:

- `int execl(char* ref, char* argv0, ..., char* argvN)`
- `int execv(char* ref, char* argv[])`
  - *ref* = argument obligatoriu, fiind numele procesului care va reacoperi procesul apelant al respectivei primitive `exec`
  - celelalte argumente pot lipsi; ele exprimă parametrii liniei de comandă pentru procesul *ref*

Note: i) argumentul *ref* trebuie să fie un nume de fişier executabil care să se afle în directorul curent (sau să se specifice şi directorul în care se află, prin cale absolută sau relativă), deoarece nu este căutat în directoarele din variabila de mediu `PATH`.
ii) ultimul argument *argvN*, respectiv ultimul element din tabloul *argv*`[]`, trebuie să fie pointerul `NULL`.

Primitivele din familia `exec` (cont.)

2) A doua pereche de primitive `exec` este perechea `execle` şi `execve`, cu interfeţele următoare:

- `int execle(char* ref, char* argv0, ..., char* argvN, char* env[])`
- `int execve(char* ref, char* argv[], char* env[])`
  - *env* = parametru ce permite transmiterea către noul proces a unui *environment* (*i.e.*, un set de variabile de mediu)
  - celelalte argumente sunt la fel ca la prima pereche

Note: i) şi în acest caz, *ref* trebuie să fie un nume de fişier executabil care să se afle în directorul curent (sau să se specifice şi directorul în care se află, prin cale absolută sau relativă), deoarece nu este căutat în directoarele din variabila de mediu `PATH`.
ii) la fel ca pentru *argv*`[]`, ultimul element din tabloul *env*`[]` trebuie să fie pointerul `NULL`.

Primitivele din familia `exec` (cont.)

3) A treia pereche de primitive `exec` este perechea `execlp` şi `execvp`, cu interfeţele următoare:

- `int execlp(char* ref, char* argv0, ..., char* argvN)`
- `int execvp(char* ref, char* argv[])`
  - argumentele sunt la fel ca la prima pereche

Notă: argumentul *ref* indică un nume de fişier executabil ce este căutat în directoarele din variabila de mediu PATH, în cazul în care nu este specificat împreună cu calea, relativă sau absolută, până la acel fişier.

Primitivele din familia `exec` (cont.)

Valoarea returnată: în caz de eşec (datorită memoriei insuficiente, sau altor cauze), toate primitivele `exec` returnează valoarea `-1`.
Altfel, în caz de succes, *apelul `exec` nu returnează* (!), deoarece procesul apelant nu mai există (a fost reacoperit de noul proces).
Notă: `exec` este singurul exemplu de funcţie (cu excepţia primitivei `exit`) al cărei apel nu returnează înapoi în programul apelant.

Primitivele din familia `exec` (cont.)

Valoarea returnată: în caz de eşec (datorită memoriei insuficiente, sau altor cauze), toate primitivele `exec` returnează valoarea `-1`.
Altfel, în caz de succes, *apelul `exec` nu returnează* (!), deoarece procesul apelant nu mai există (a fost reacoperit de noul proces).
Notă: `exec` este singurul exemplu de funcţie (cu excepţia primitivei `exit`) al cărei apel nu returnează înapoi în programul apelant.

Observaţie: prin convenţie *argv0*, respectiv *argv*`[0]`, trebuie să coincidă cu *ref* (deci cu numele fişierului executabil). Aceasta este însă doar o *convenţie*, nu se produce eroare în caz că este încălcată.
De fapt, argumentul *ref* specifică *numele real* al fişierului executabil ce se va încărca şi executa, iar *argv0*, respectiv *argv*`[0]`, specifică *numele afişat* (de comenzi precum `ps`, `w`, ş.a.) al noului proces.

Caracteristicile procesului după `exec`

Noul proces moşteneşte caracteristicile vechiului proces (are acelaşi `PID`, aceeaşi prioritate, acelaşi proces părinte, aceeaşi descriptori de fişiere deschise, etc.), cu unele excepţii, în condiţiile specificate:

Caracteristica	**Condiţia în care nu se conservă**
Proprietarul efectiv	Dacă este setat bitul *setuid* al fişierului încărcat, proprietarul acestui fişier devine proprietarul efectiv al procesului
Grupul efectiv	Dacă este setat bitul *setgid* al fişierului încărcat, grupul proprietar al acestui fişier devine grupul proprietar efectiv al procesului
Handler-ele de semnale	Sunt reinstalate *handler*-ele implicite pentru semnalele corupte
Descriptorii de fişiere	Dacă bitul `FD_CLOEXEC` de închidere automată în caz de `exec`, al vreunui descriptor de fişier, a fost setat cu ajutorul primitivei `fcntl`, atunci descriptorul respectiv este închis la `exec` (ceilalţi descriptori de fişiere rămân deschişi)

Exemplu (1): Reacoperirea procesului

Un exemplu ce ilustrează câteva dintre aceste proprietăţi:

a se vedea programul `before_exec.c` , ce apelează `exec` pentru a se reacoperi cu un al doilea program, `after_exec.c` .

Ca o dovadă a faptului că noul proces `after_exec` moşteneşte descriptorii de fişiere deschise de la procesul `before_exec`, în urma execuţiei veţi constata că variabila `nrBytesRead` are valoarea `-1`, motivul fiind că intrarea standard `stdin` este moştenită ca fiind închisă în procesul `after_exec`.

Exemplu (2): Reacoperire recursivă

Un al doilea exemplu: un program care se reacoperă cu el însuşi, dar la al doilea apel îşi modifică parametrii de apel pentru a-şi putea da seama că este la al doilea apel şi astfel să nu intre într-un apel recursiv la infinit.

A se vedea programul `exec_rec.c` .

Exemplu (3): Reacoperire cu fişiere deschise

A se vedea programul `com-0.c` care se reacoperă cu `com-2.c` .

Observaţie: programul `com-0` redirectează fluxul `stdout` în fişierul `fis.txt`, şi ca atare `com-2` moşteneşte această redirectare; mesajele scrise vor apare în acel fişier şi nu pe ecran.

Exemplu (3): Reacoperire cu fişiere deschise

A se vedea programul `com-0.c` care se reacoperă cu `com-2.c` .

Observaţie: programul `com-0` redirectează fluxul `stdout` în fişierul `fis.txt`, şi ca atare `com-2` moşteneşte această redirectare; mesajele scrise vor apare în acel fişier şi nu pe ecran.

Comportamentul în cazul fişierelor deschise în momentul apelului primitivelor `exec`: dacă s-au folosit instrucţiuni de scriere *buffer*-izate (ca de exemplu funcţiile `fprintf`, `fwrite` ş.a. din biblioteca standard I/O din C), atunci *buffer*-ele nu sunt scrise automat în fişier pe disc în momentul apelului `exec`, deci informaţia din ele se pierde.

Notă: în mod normal *buffer*-ul este scris în fişier abia în momentul când s-a umplut, sau la întâlnirea caracterului '\n'. Dar se poate forţa scrierea *buffer*-ului în fişier cu ajutorul funcţiei `fflush` din biblioteca standard I/O din C.

Exemplu (3): Reacoperire cu fişiere deschise

A se vedea programul `com-0.c` care se reacoperă cu `com-2.c`.

Observaţie: programul `com-0` redirectează fluxul `stdout` în fişierul `fis.txt`, şi ca atare `com-2` moşteneşte această redirectare; mesajele scrise vor apare în acel fişier şi nu pe ecran.

A se vedea programul `com-1.c` care se reacoperă cu `com-2.c`.

Observaţie: dacă eliminăm apelul `fflush` din programul `com-1.c`, atunci pe ecran se va afişa doar mesajul "`la toti!`"; mesajul "`Salut`" se pierde prin `exec`, *buffer*-ul nefiind golit pe disc.

Exemplu (4): Redirectarea fluxului `stdout`

Pe lângă primitiva `dup`, mai există o primitivă, cu numele `dup2`, utilă pentru duplicarea unui descriptor de fişier. Cu ajutorul lor se poate realiza redirectarea fluxurilor standard de I/O, aşa cum am văzut în exemplul precedent (*i.e.*, programul `com-0.c`).

Un alt exemplu de redirectare a fluxului `stdout`:
A se vedea programul `redirect.c` .

În acest caz redirectarea se face către fişierul `fis.txt`, iar apoi este anulată (prin redirectarea înapoi către terminalul I/O fizic, `/dev/tty`).

Bibliografie obligatorie

Cap.4, §4.4 din manualul, în format PDF, accesibil din pagina disciplinei “Sisteme de operare”:

- `http://profs.info.uaic.ro/~vidrascu/SO/books/ManualID-SO.pdf`

Programele demonstrative amintite pe parcursul acestei prezentări pot fi descărcate de la adresa următoare:

- `http://profs.info.uaic.ro/~vidrascu/SO/cursuri/C-programs/exec/`