Tema 4 Planificator de threaduri

Dată publicare: **27 Aprilie 2017**

Deadline: **10 Mai 2017, ora 23:55**

Deadline hard: **17 Mai 2017, ora 23:55**

Obiectivele temei

- Aprofundarea următoarelor noțiuni:
  - threaduri POSIX și WINDOWS
  - planificarea proceselor/threadurilor
  - mecanisme de sincronizare între threaduri

Enunț

Să se implementeze un planificator de threaduri care va controla execuția acestora în user-space. Acesta va simula un **scheduler** de procese preemptiv, într-un sistem uniprocesor, care utilizează un algoritm de planificare Round Robin cu priorități.

Implementarea planificatorului de threaduri se va face într-o bibliotecă partajată dinamică, pe care o vor încărca thread-urile ce urmează să fie planificate. Aceasta va trebui să exporte următoarele funcții:

- `INIT` - inițializează structurile interne ale planificatorului.
- `FORK` - pornește un nou thread.
- `EXEC` - simulează execuția unei instrucțiuni.
- `WAIT` - așteaptă un eveniment/operație I/O.
- `SIGNAL` - semnalează threadurile care așteaptă un eveniment/operație I/O.
- `END` - distruge planificatorul și eliberează structurile alocate.

Pe lângă implementarea propriu-zisă a funcțiilor de mai sus, va trebui să asigurați și planificarea/execuția corectă a threadurilor, conform algoritmului Round Robin cu priorități. Fiecare thread trebuie să ruleze în contextul unui thread real din sistem. Fiind un planificator pentru sisteme uniprocesor, un singur thread va putea rula la un moment dat.

Descriere detaliată a funcționării planificatorului

Timp execuție:

Într-un sistem real, pentru controlul execuției, contorizarea timpului de rulare a unui proces se realizează la fiecare întrerupere de ceas.

Pentru facilitarea implementării, modelul temei va simula un sistem real astfel:

1. sistemul simulat va folosi un timp virtual (logic), independent de cel real pentru a contoriza timpul de rulare pe procesor.
2. veți considera că o `instructiune` durează o **singură** perioadă de ceas (unitate de timp logic).
3. fiecare din funcțiile prezentate mai sus reprezintă o **singură** `instructiune` ce poate fi executată de un thread la un moment

Search

Informații generale SO

- Documentație și alte resurse
- Feed RSS
- Hall of SO
- Listă de discuții
- Mașini virtuale
- Trimitere teme

Informații SO 2016-2017

- **Examen**
  - Examen CA/CC 2012-2013
  - Examen CA/CC 2013-2014
  - Examen CA/CC 2014-2015
  - Examen CA/CC 2015-2016
- **Reguli generale și notare**
  - Notare CA/CB/CC
- Anunțuri
- Calendar
- Catalog
- Echivalări teme
- Karma Awards
- SO Need to Know
- Orar și împărțire pe semigrupe

Laboratoare

- **Resurse**
  - C/SO Tips
  - Macro-ul DIE
  - GDB
  - Resurse
  - Function Hooking and Windows Dll Injection
  - Oprofile
  - Recapitulare
  - Thread-uri - Extra
  - Visual Studio Tips and Tricks
  - windows-video
- Laborator 01 - Introducere
- Laborator 02 - Operații I/O simple
- Laborator 03 - Procese
- Laborator 04 - Semnale
- Laborator 05 - Gestiunea memoriei

dat.

Evenimente și I/O

Threadurile din sistem se pot bloca în așteptarea unui eveniment sau a unei operații de I/O. Un astfel de eveniment va fi identificat printr-un id (0 - nr_events). Numărul total de evenimente (care pot apărea la un moment dat în sistem) va fi dat ca parametru la inițializarea planificatorului.

Un thread se blochează în urma apelului funcției `wait` (ce primește ca parametru id-ul/indexul evenimentului/dispozitivului) și este eliberat atunci când un alt thread apelează funcția `signal` pe același eveniment/dispozitiv. Signal trezește toate threadurile care așteapta un anumit eveniment.

Atenție, un thread care a apelat funcția `wait`, indiferent de prioritate, nu poate fi planificat decât după ce este trezit de un alt thread.

Round Robin cu priorități

- fiecare task are asociată o prioritate statică, care este specificată la pornirea threadului.
- întotdeauna va fi planificat threadul cu cea mai mare prioritate.
- la expirarea cuantei de timp, threadul care rulează este preemptat și se alege un nou task.
- dacă un thread este preemptat și revine în starea READY (ex: cazul în care îi expiră cuanta), iar în sistem nu mai există un alt thread READY de prioritate mai mare sau egală cu a lui, va fi replanificat același thread.
- threadurile cu aceeași prioritate vor fi planificate după modelul Round Robin (ex: puteți folosi cozi de priorități ce vor fi parcurse circular).
- dacă un thread proaspăt creat (sau care doar ce a ieșit din starea de waiting) are prioritate mai mare decât threadul care rulează, cel din urmă va fi preemptat și va rula thread-ul cu prioritatea mai mare.
- la replanificarea unui thread, cuanta acestuia de rulare pe procesor va fi resetată la valoarea maximă.
- **preempția**: odată ce o funcție/instrucțiune a fost planificată, ea nu poate fi preemptată decât după ce și-a terminat treaba ("do work" în exemplul de mai jos). Pentru a simula preempția, puteți bloca funcția curentă folosind mecanisme de sincronizare. De exemplu, o instrucțiune se poate abstractiza astfel:

```
instruction()
{
    do work
    check scheduler
    if (preempted)
        block();
    return;
}
```

În momentul în care threadul a fost planificat pe procesor, se va executa secvența de cod de mai sus. Funcția "do work" va fi executată imediat ce threadul este planificat. Este nevoie să verificăm dacă threadul a fost preemptat, iar acest lucru se verifică in funcția "check scheduler". Dacă "check scheduler" va semnala că threadul a fost preemptat, atunci acesta se va bloca până când va fi planificat din nou.

Prin urmare, cazurile în care threadul curent este preemptat și un alt thread începe rularea sunt:

- un task cu o prioritate mai mare intră în sistem.
- un task cu o prioritate mai mare este semnalat printr-o operație de tipul `signal`.
- task-ului curent i-a expirat cuanta de timp.
- așteaptă la un eveniment cu ajutorul unei operații de tipul `wait`.
- nu mai are instrucțiuni de executat.

Pentru mai multe detalii despre algoritmi de planificare puteți consulta

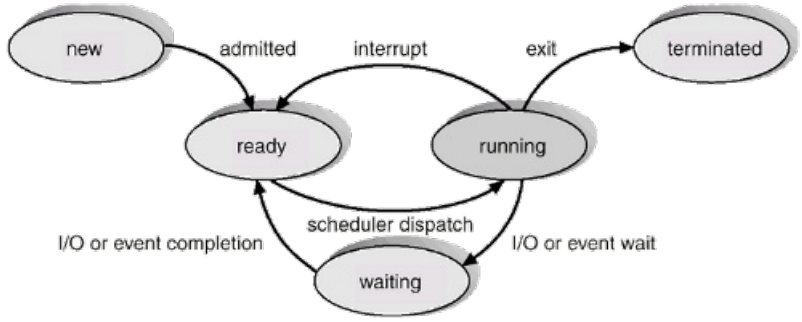
cursul 4.

Stări threaduri:

Stările prin care poate trece un thread sunt:

- **New** - thread nou creat în urma unui apel de tipul `fork`.
- **Ready** - așteaptă să fie planificat.
- **Running** - planificat - un singur thread poate rula la un moment dat.
- **Waiting** - așteaptă după un eveniment sau o operație I/O. Un thread se va bloca în starea de wait în urma apelului `so_wait(event/io)`.
- **Terminated** - și-a încheiat execuția.

Pentru o mai bună întelegere a algoritmilor de planificare, se recomandă urmărirea tranzițiilor dintre stări ca în desenul de mai jos.



Detalii implementare instrucțiuni

Funcțiile care trebuie exportate de planificator, alături de parametrii fiecăruia, sunt detaliate mai jos:

- `int so_init(cuantă, io)` - inițializează planificatorul. Primește ca argumente cuanta de timp după care un proces trebuie preemptat și numărul de evenimente (dispozitive I/O) suportate. Întoarce 0 dacă planificatorul a fost inițializat cu succes, sau negativ în caz de eroare. Numărul maxim de dispozitive I/O suportate este 256 iar indexarea acestora începe de la 0.
- `void so_end()` - eliberează resursele planificatorului și așteaptă terminarea tuturor threadurilor înainte de părăsirea sistemului.
- `void so_exec()` - simulează execuția unei instrucțiuni generice. Practic, doar consumă timp pe procesor.
- `int so_wait(event/io)` - threadul curent se blochează în așteptarea unui eveniment sau a unei operații de I/O. Întoarce 0 dacă evenimentul există (id-ul acestuia este valid) sau negativ în caz de eroare.
- `int so_signal(event/io)` - trezește unul sau mai multe threaduri care așteaptă un anumit eveniment. Întoarce numărul total de threaduri deblocate, sau negativ în caz de eroare (evenimentul nu este valid).
- `tid_t so_fork(handler, prioritate)` - pornește și introduce în planificator un nou thread. Primește ca parametru o rutină pe care threadul o va executa după ce va fi planificat și prioritatea cu care acesta va rula și întoarce un id unic corespunzător threadului. Handlerul executat de thread va primi ca parametru prioritatea acestuia.

În mod normal, so_fork va fi apelată din contextul unui alt thread din sistem. Se garantează faptul că va exista întotdeauna cel puțin un thread ce poate fi planificat, pe întreg parcursul rulării planificatorului. Excepție face cazul primului so_fork ce va crea primul thread din sistem și va fi apelat din contextul testelor, neavând ca parinte un thread din sistemul simulat.

Un exemplu de model de implementare a funcției fork, ar putea fi folosirea unei funcții suplimentare (ex start_thread) care să determine contextul în care se va executa noul thread (handlerul primit ca parametru) ca în descrierea de mai jos:

```
so_fork(handler, prio)
-> initializare thread struct
-> creare thread nou ce va executa functia start_thread
-> asteaptă ca threadul să intre în starea READY/RUN
-> returnează id-ul noului thread

start_thread(params)
-> asteaptă să fie planificat
-> call handler(prio)
-> iesire thread
```

Atenție: Funcția se va întoarce abia după ce noul thread creat fie a fost planificat fie a intrat în starea READY.

Exemplu execuție

= Thread 0 =	= Thread 1 =	= Thread 2 =	= Thread 3 =
exec	exec	exec	exec
fork(thr2, 2)	signal(3)	wait(3)	exec
fork(thr1, 1)	fork(thr3, 1)	exec	
exec	exec		
	exec		
	exec		

În exemplul de mai sus, threadul 0 are prioritatea 0. Acesta pornește threadurile 1 și 2 cu prioritățile asociate. Threadul 2 se blochează așteptând evenimentul cu id-ul 3, eveniment ce va fi semnalat de threadul 1. Cuanta de rulare pe procesor este 3. În final, instrucțiunile se vor executa în următoarea ordine:

```
T0 exec
T0 forks T2, prio 2
--> T2 preempts T0 (prio(T2) > prio(T0))
T2 exec
T2 waits for event 3
--> T2 blocks and is preempted
T0 forks T1, prio 1
--> T1 preempts T0
T1 exec
T1 signals event 3
--> T2 is woken up and preempts T1
T2 exec
--> T2 finished
T1 forks T3, prio 1
T1 exec
T1 exec
--> T1 is preempted, its CPU quantum expired
T3 exec
T3 exec
--> T3 finished
T1 exec
--> T1 finished
T0 exec
--> T0 finished
```

Precizări Linux

Identificatorul întors de funcția `so_fork` trebuie să fie structura `pthread_t` populată de funcția POSIX `pthread_create()`.

Tema se va rezolva folosind doar funcții POSIX. Se pot folosi de asemenea și funcțiile de formatare printf, scanf, funcțiile de alocare de memorie malloc, free, și funcțiile de manipulare a șirurilor de caractere (strcat, strdup, etc.)

Tema se va rezolva folosind fire de execuție POSIX și exclusiv mecanisme de sincronizare a firelor de execuție POSIX (mutex, variabile de condiție).

Precizări Windows

Pe lângă mecanismele de sincronizare învățate la laborator, aveți voie să folosiți și variabile de condiție.

Identificatorul întors de funcția `so_fork` trebuie să fie cel întors de funcția `GetCurrentThreadId()` apelată în contextul threadului nou creat.

Tema se va rezolva folosind doar funcții Win32. Se pot folosi de asemenea și funcțiile de formatare printf, scanf, funcțiile de alocare de memorie malloc, free și funcțiile de manipulare a șirurilor de caractere (strcat, strcmp, etc.)

Utile

- Header - conține semnăturile funcțiilor exportate.

Testare

- Corectarea temelor se va realiza automat cu ajutorul unor suite de teste publice:
  - teste Linux – tema4-checker-lin.zip
  - teste Windows – tema4-checker-win.zip
- Pentru evaluare și corectare tema va fi uploadată folosind interfața vmchecker.
- În urma compilării temei trebuie să rezulte o bibliotecă shared-object (Linux) denumită `libscheduler.so` sau o bibliotecă dinamică (Windows) denumită `libscheduler.dll`.
- Suita de teste conține un set de teste. Trecerea unui test conduce la obținerea punctajului aferent acestuia.
  - În urma rulării testelor, se va acorda, în mod automat, un punctaj total. Punctajul total maxim este de 90 de puncte, pentru o temă care trece toate testele. La acest punctaj se adaugă 10 puncte care reprezintă aprecierea temei de către asistentul care o corectează.
  - Cele 100 de puncte corespund la 10 puncte din cadrul notei finale.
- Pot exista penalizări în caz de întârzieri sau pentru neajunsuri de implementare sau de stil. Ca excepție, pot apărea depunctări mai mari de 1 pct, în cazul în care tema trece teste prin "hackuri" în implementare (ex: sincronizări cu sleep, threadurile nu rulează peste threaduri reale din sistem, etc).
- **Testul 0** din cadrul checker-ului temei verifică automat coding style-ul surselor voastre. Ca referință este folosit stilul de coding din kernelul Linux. Acest test valorează 5 puncte din totalul de 100. Pentru mai multe informații despre un cod de calitate citiți pagina de recomandări.

Înainte de a uploada tema, asigurați-vă că implementarea voastră trece testele pe mașinile virtuale. Dacă apar probleme în rezultatele testelor, acestea se vor reproduce și pe vmchecker.

Notare

Nota maximă este de 100p.

Depunctări suplimentare:

- `-0.5`: alocare statică a unui vector de thread-uri

- `-0.6`: erori de sincronizare (deadlock, race condition, etc.)
- `-2`: sincronizare prin sleep-uri
- `-2`: sincronizare prin busy-waiting
- `-1`: nu sunt definite structuri pentru thread-uri, planificator, cozi de priorități; lipsa abstractizării și a încapsulării datelor
- nerespectarea regulilor de aici
- alte depunctări pentru implementări greșite, ce nu respectă cerința generală a temei

Materiale ajutătoare

Cursuri:

- Curs 4 - Planificarea execuției
- Curs 8 - Fire de execuție
- Curs 9 - Sincronizare

Laboratoare:

- Laborator 8
- Laborator 9

Resursele temei se găsesc și în repo-ul so-assignments de pe GitHub. Repo-ul conține și un script Bash care vă ajută să vă creați un repository privat pe instanța de GitLab a facultății. Urmăriți indicațiile din README și de pe pagina de Wiki dedicată pentru git.

În plus, responsabilii de teme se pot uita mai rapid pe GitLab la temele voastre în cazul în care aveți probleme/bug-uri. Este mai ușor să primiți suport în rezolvarea problemelor implementării voastre dacă le oferiți responsabililor de teme acces la codul sursă pe GitLab.

Dacă ați folosit GitLab pentru realizarea temei, indicați în README link-ul către repository. Asigurați-vă că responsabilii de teme au drepturi de citire asupra repo-ului vostru.

FAQ

- **Q:** Tema se poate face în C++?
  - **A:** Nu.

Suport, întrebări și clarificări

Pentru întrebări sau nelămuriri legate de temă folosiți lista de discuții sau canalul de IRC.