Laborator 8

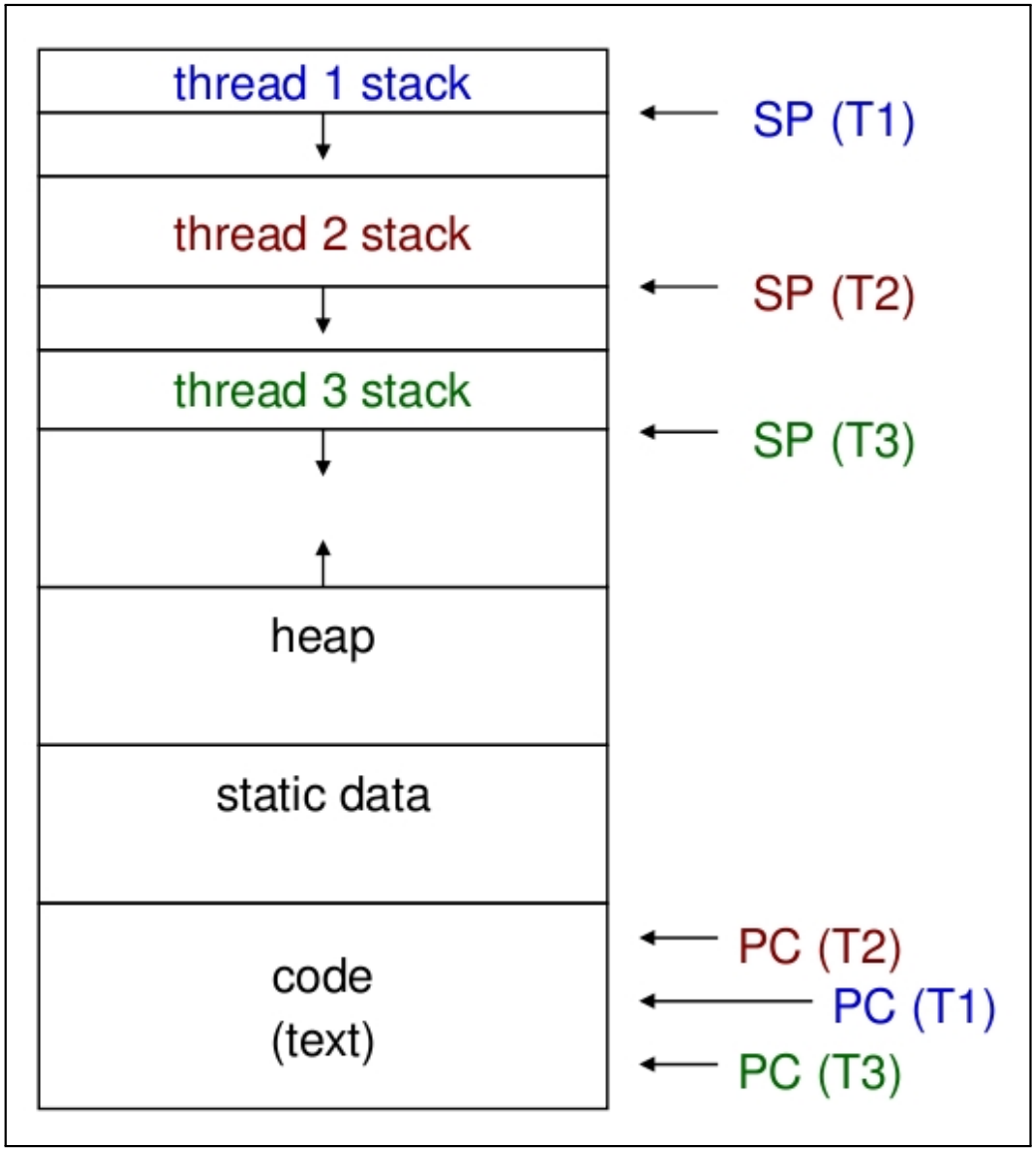
Thread-uri

Sisteme de Operare

7 - 13 aprilie 2011

Thread Control Block

- Informații partajate
  - Spațiu de adresă
  - Heap, Data
  - Semnale și handlere
  - I/O and file
- Informații proprii
  - Starea
  - Regiștrii
  - Program counter
  - Stiva
  - Masca de semnale
  - errno



Avantaje

- pot comunica între ele fără a implica kernelul
- asigură o folosire mai eficientă a resurselor calculatorului
- mai puțin timp pentru crearea/distrugerea unui thread decât a unui proces
- comutarea între 2 threaduri mai rapidă decât între procese
- Paralelizarea are sens in 2 situații:
  - task-uri I/O bound pe același CPU
  - task-uri CPU bound pe mai multe core-uri
- dezavantaj: sincronizare (overhead + model de programare complex)

Thread-uri – noțiuni generale

- Nu întotdeauna dorim să partajăm totul cu celelalte thread-uri => e nevoie de un storage thread-specific
- O zonă din stiva fiecărui thread este organizată sub forma unui Map cu perechi (cheie, valoare)
- Atenție la crearea de foarte multe thread-uri cu stiva mare (se poate epuiza spațiul de adrese)

- Există 3 tipuri de implementări : ULT, KLT, hibride

ULT vs. KLT

- **User-Level Threads**
  - Kernel-ul nu este conștient de existența lor
  - Schimbarea de context nu implică kernelul => rapidă
  - Planificarea poate fi aleasă de aplicație
  - Aceste thread-uri pot rula pe orice SO
  - Dacă un thread apelează ceva blocant toate thread-urile planificate de aplicație vor fi blocate
  - 2 fire ale unui proces nu pot rula simultan pe 2 procesoare
- **Kernel-Level Threads**
  - Schimbarea de context între thread-uri ale aceluiași proces implică kernel-ul => viteza de comutare este mică
  - Blocarea unui fir nu înseamnă blocarea întregului proces
  - Dacă avem mai multe procesoare putem lansa în execuție simultană mai multe thread-uri ale aceluiași proces

- **Thread-safe** - Operații sigure în context multithreading
  - o funcție este thread-safe dacă și numai dacă va produce mereu rezultatul corect atunci când este apelată concurent, în mod repetat din mai multe threaduri
- Tipuri de funcții thread-unsafe
  - Funcții ce nu protejează variabilele partajate
  - Funcții ce nu păstrează starea la apeluri multiple
  - Funcții ce întorc pointer la o variabilă statică
  - Funcții ce apelează funcții thread-unsafe
- Funcțiile **reentrante** sunt cele care nu referă date partajate
  - nu lucrează cu variabile globale/statice
  - nu apelează funcții non-reentrante
  - sunt un subset al funcțiilor thread-safe
- un apel reentrant în execuție nu afectează un alt apel simultan

- Mutex (POSIX, Win32)
- Semafor (POSIX, Win32)
- Secțiune critică (Win32)
- Variabilă de condiție (POSIX)
- Barieră (POSIX)
- Operații atomice cu variabile partajate (Win32)
- Thread pooling (Win32)

Întrebări

- Cum se modifică spațiul de adresă al unui proces la schimbarea de context între două fire de execuție?
- Un proces are 4 fire de execuție. Unul din firele de execuție, face un access invalid la memorie. Ce se întâmplă cu celelalte 3 fire de execuție?
- Descrieți o situație în care folosirea firelor de execuție ar duce la degradarea performanței?
- Descrieți o situție (pe un sistem uniprocesor) în care folosind fire de execuție îmbunătățim performanța unui program.