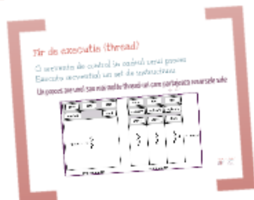
Fire de Executie

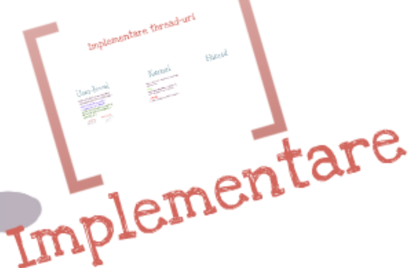
Sisteme de Operare, Curs 8

Thread-uri



Server de Web

API



Implementare

Sincronizare

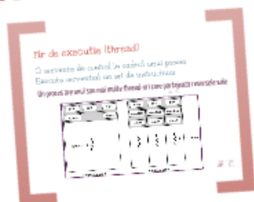
Gata!

Prezi

Fire de Executie

Sisteme de Operare, Curs 8

Thread-uri



Server de Web

API

Implementare

Sincronizare

Gata!

PREZI

Cum implementam un server de web?

Cerinte

Serveste un numar arbitrar de clienti
Fiecare client poate cere oricate pagini (HTTP 1.1)
Mentine statistici: nr. total de pagini vizionate, numarul total de clienti, octeti cititi, etc.

Alternative

Implementare Secventiala

```
while (1){
    int s = accept(ls);
    fname = read_request(s);
    while (fname){
        read_and_send_file(fname);
        update_stats();
        fname = read_request(s);
    }
}
```

Folosind Procese

```
while (1){
    int s = accept(ls);
    if (fork()==0){
        fname = read_request(s);
        while (fname){
            read_and_send_file(fname);
            update_stats();
            fname = read_request(s);
        }
    } else {...}
}
```

Implementare Asincrona

Folosind Thread-uri

Prezi

Implementare Secventiala

```
while (1){
        int s = accept(ls);
        fname = read_request(s);
        while (fname){
                read_and_send_file(fname);
                update_stats();
                fname = read_request(s);
        }
}
```

Probleme

Un singur client simultan
Ineficient chiar si cu un singur procesor

Prezi

Probleme

Un singur client simultan

Ineficient chiar si cu un singur procesor

Prezi

```
while (1){
	int s = accept(ls);
	fname = read_request(s);
	while (fname){
		read_and_send_file(fname);
		update_stats();
		fname = read_request(s);
	}
}
```

Probleme

Un singur client simultan
Ineficient chiar si cu un singur procesor

Prezi

Alternative

losind Procese

```
1){
t s = accept(ls);
(fork()==0){
  fname = read_request(s);
```

Prezi

Implementare Asincrona

```
while (1){
    int s = accept(ls);
    add_client(s);
    select(...);
    for (c:clients){
        if (FD_ISSET(c.s)){
```

FoloSind ProceSe

```
while (1){
    int s = accept(ls);
    if (fork()==0){
        fname = read_request(s);
        while (fname){
            read_and_send_file(fname);
            update_stats();
            fname = read_request(s);
        }
    } else { ... }
}
```

Probleme

Cum actualizam statisticile?
Cost mare pentru pornire proces

Prezi

Probleme

Cum actualizam statisticile?
Cost mare pentru pornire proces

Prezi

Implementare Asincrona

```
while (1){
    int s = accept(ls);
    add_client(s);
    select(...);
    for (c:clients){
        if (FD_ISSET(c.s)){
            fname = read_request(s);
            c.d = open(fname,...);
            c.status = read_file;
            //...
        } else if (FD_ISSET(c.d)){
            read(c.d, buf, 1000);
            send(c.s,buf,1000);
            ...
        }
    }
}
```

Probleme

Trebuie sa tinem stare pentru fiecare client
Greu de implementat

Prezi

Probleme

Trebuie sa tinem stare pentru fiecare client
Greu de implementat

Prezi

Am dori o primitiva SO care:

Executa secvential un set de intructiuni

Este usor de pornit / oprit

Partajeaza date cu usurinta

Prezi

FoloSind Thread-uri

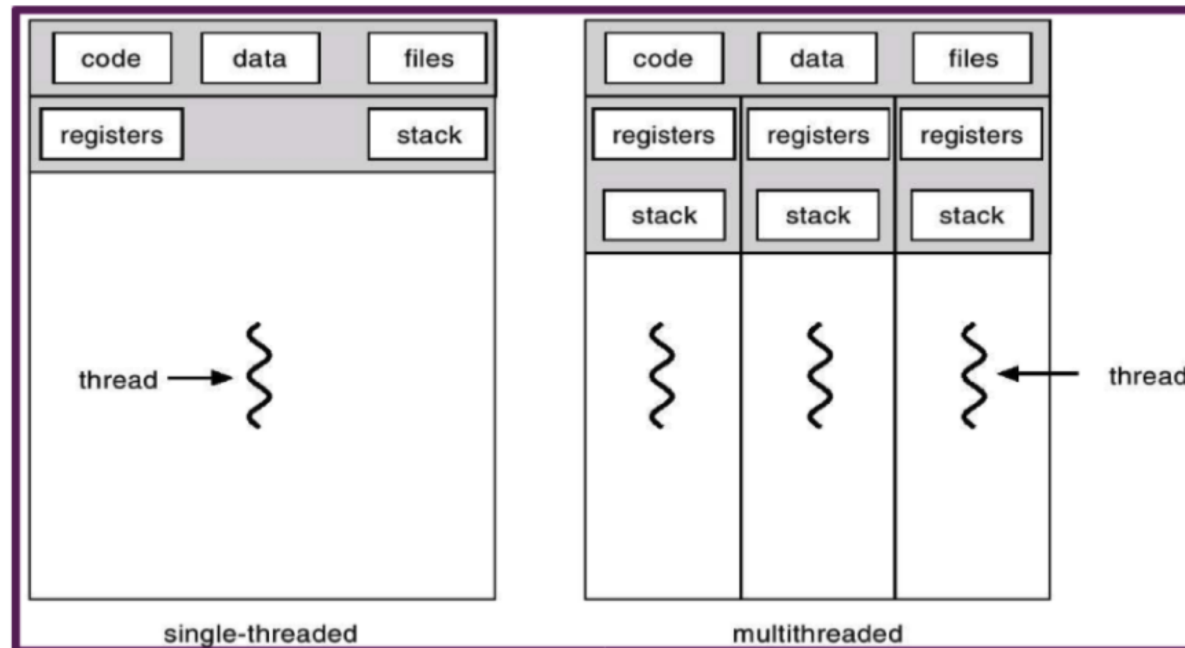
```
while (1){
        int s = accept(ls);
        pthread_create (&t, NULL,(void *) &cnt, (void *) &s);
}
...
void* clnt(void* p){
        int s = *(int*)p;
        char* fname = read_request(s);
        while (fname){
                read_and_send_file(fname);
                update_stats();
                fname = read_request(s);
        }
}
```

Prezi

Fir de executie (thread)

O secventa de control în cadrul unui proces
Executa secvential un set de instructiuni

Un proces are unul sau mai multe thread-uri care partajeaza resursele sale



Prezi

Ce partajeaza thread-urile?

variabilele globale (.data, .bss)

fisierele deschise

spatiul de adresa

masca de semnale

Prezi

Ce NU partajeaza thread-urile?

- registrele
- stiva
- program counter/Instruction pointer
- stare
- TLS (Thread Local Storage)

Prezi

Procese vs. thread-uri

Grupeaza resurse

- Fisiere, lucru retea
- Spatiu adrese
- Fire de executie

Abstractizeaza executia

- Stiva
- Registri
- Program Counter

Prezi

Avantaje thread-uri

Timp de creare mai mic decat al proceselor

Timp mai mic de schimbare context

Partajare facila de informatie

Utile chiar si pe uniprocesor

Prezi

Dezavantaje thread-uri

Daca moare un thread, moare tot procesul

Nu exista protectie la partajarea datelor

Probleme de sincronizare

Prea multe thread-uri afecteaza performanta!

Prezi

Operatii cu thread-uri

- Lansarea in executie
- Incetarea executiei
- Terminare fortata (cancel)
- Asteptare (join)
- Planificare

Posix Threads

Linux

Windows

Prezi

Posix Threads

Folosit pe sistemele Unix

API pentru crearea si sincronizarea thread-urilor

Folosire

- inclus header-ul (#include <pthread.h>)
- legarea bibliotecii (-lpthread)
- man 7 pthreads

Prezi

API PThreads

```
pthread_t tid;
pthread_create(&tid, NULL, threadfunc, (void*)arg);

pthread_exit(void* ret);

pthread_join(pthread_t tid, void** ret);

pthread_cancel(pthread_t tid);
```

Prezi

Thread-uri in Linux

Suport in kernel pentru task-uri (struct task_struct)
Procesele si thread-urile sunt task-uri
planificabile independent

NPTL (New Posix Thread Library)

- implementare pthreads (1:1)
- foloseste apelul de sistem clone
- thread-urile sunt grupate in acelasi grup
- getpid intoarce thread group ID

Prezi

clone

Specific Linux

Folosit de fork si NPTL

Diferite flag-uri specifica resursele partajate

- CLONE_NEWNS
- CLONE_FS, CLONE_VM, CLONE_FILES
- CLONE_SIGHAND, CLONE_THREAD

Prezi

Thread-uri in Windows

Model hibrid: suport in kernel

Fibre: fire de executie in user-mode

- planificate cooperativ
- blocarea unei fibre blocheaza firul de executie

Prezi

API Windows

HANDLE CreateThread(...)

ExitThread

WaitForSingleObject / MultipleObjects

GetExitCodeThread

TerminateThread

TlsAlloc

TlsGetValue/TlsSetValue

Prezi

Implementare thread-uri

User-level

O biblioteca de thread-uri ofera suport pentru crearea, planificarea si terminarea thread-urilor

Mentine o tabela cu fire de executie:
PC, registre, stare pentru fiecare fir

Nucleul "vede" doar procese, nu si thread-uri
Mai multe fire de executie sunt planificate cu un singur proces

Avantaje

Dezavantaje

Kernel

Suport in kernel pentru creare, terminare si planificare
Model unu-la-unu

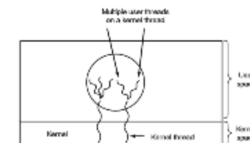
Avantaje

Fara probleme la apeluri blocante sau page faults
Pot fi planificate pe sisteme multiprocesor

Dezavantaje

Crearea si schimbarea de context este mai lenta

Hibrid



User-level

O biblioteca de thread-uri ofera suport pentru crearea, planificarea si terminarea thread-urilor

Mentine o tabela cu fire de executie:
PC, registre, stare pentru fiecare fir

Nucleul "vede" doar procese, nu si thread-uri

Mai multe fire de executie sunt planificate cu un singur proces

Avantaje

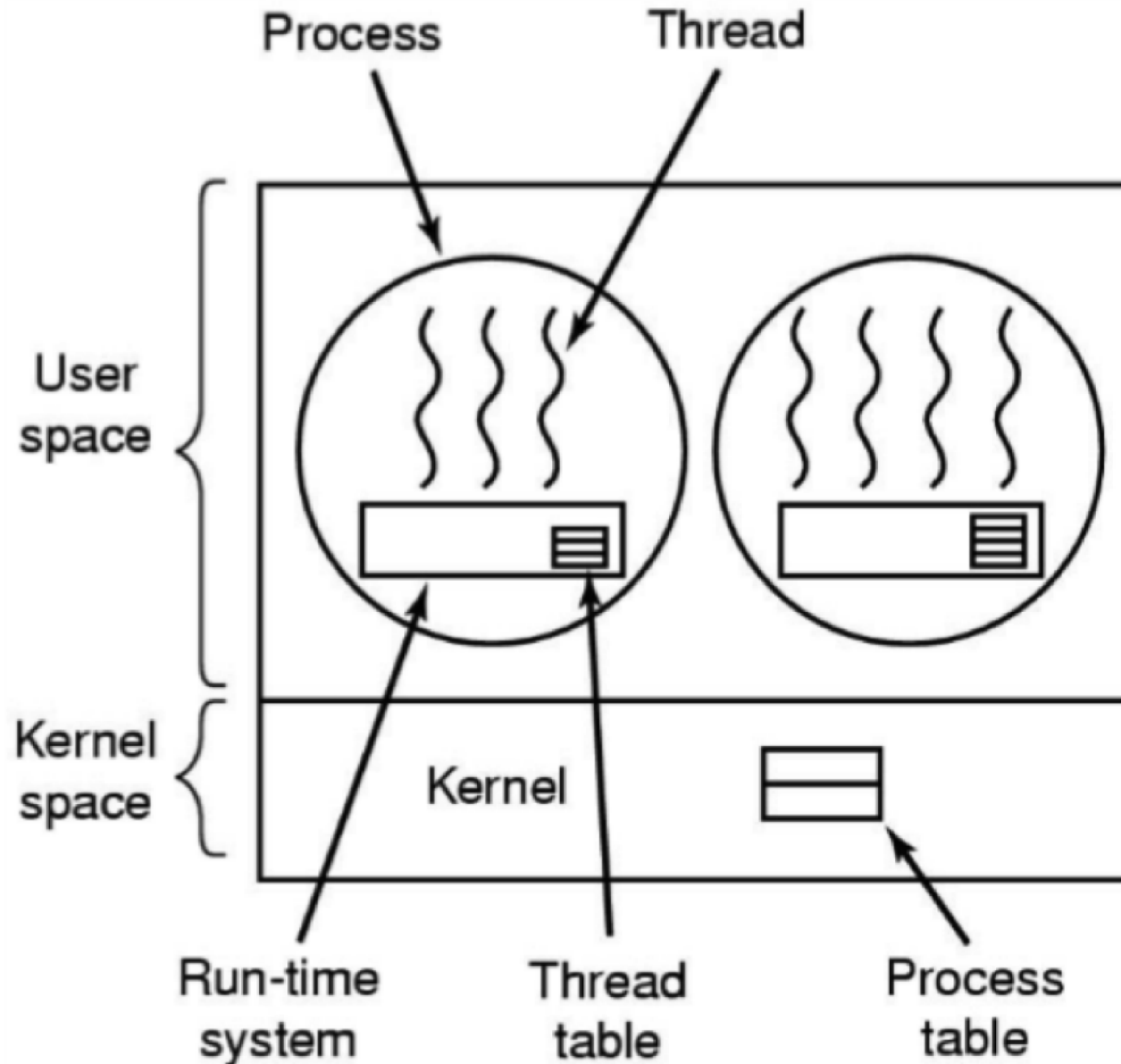
Usor de integrat în SO: nu sunt necesare modificari

Dezavantaje

Un apel de sistem blocant blocheaza întreg procesul:

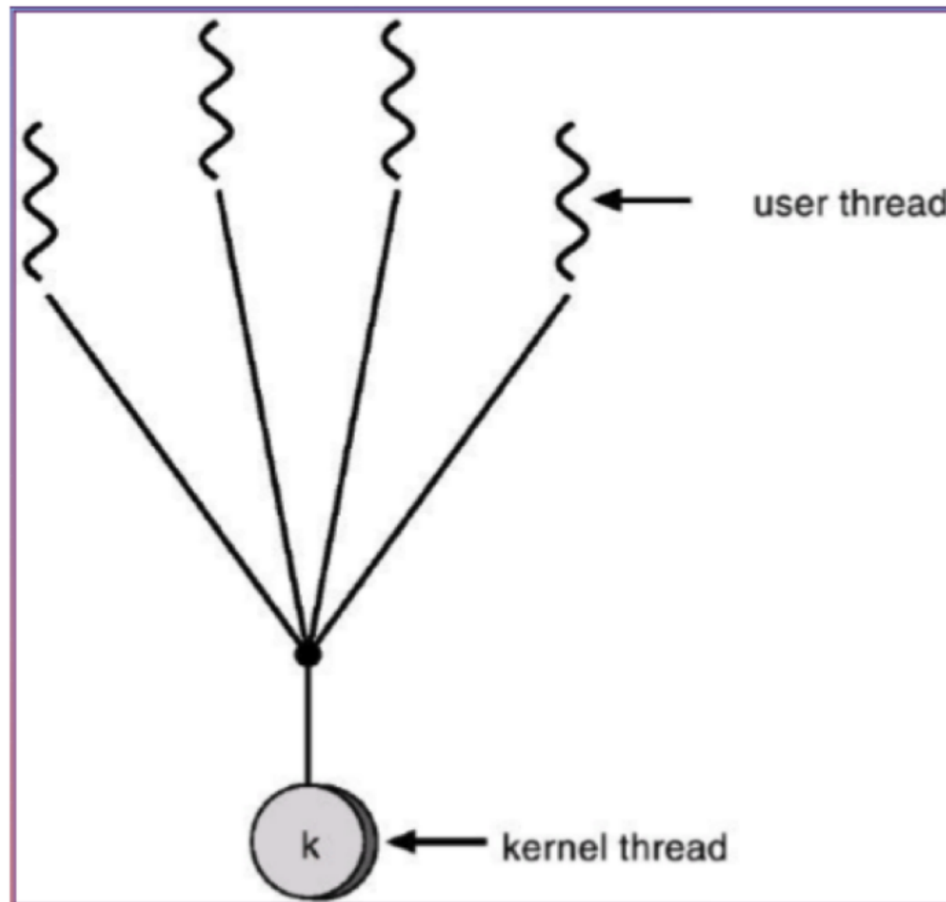
Prezi

Thread-uri implementate user-level



Prezi

Mai multe fire de executie sunt mapate pe acelasi fir de executie din kernel



Prezi

Avantaje

Usor de integrat în SO: nu sunt necesare modificari

Pot oferi suport multithreaded pe un SO fara suport multithreaded

Schimbare de context rapida: nu se executa apeluri de sistem în nucleu

Aplicatiile pot implementa planificatoare în functie de necesitati

Prezi

Dezavantaje

Un apel de sistem blocant blocheaza întreg pro

c

Un page-fault blocheaza tot procesul

Planificare cooperativa

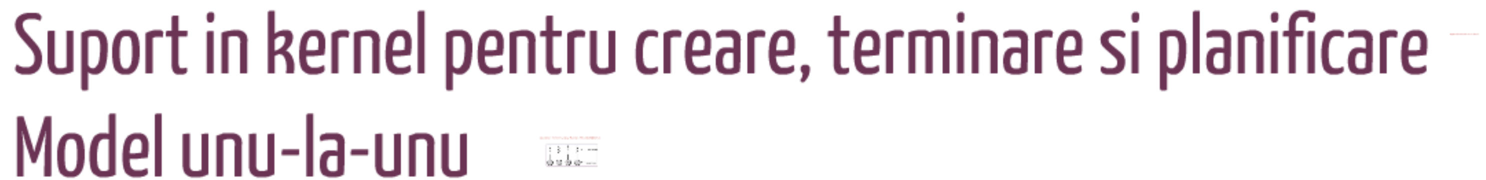
Multe aplicatii folosesc apeluri de sistem oricur

Prezi

Kernel

Suport in kernel pentru creare, terminare si planificare

Model unu-la-unu



Avantaje

Fara probleme la apeluri blocante sau page faults

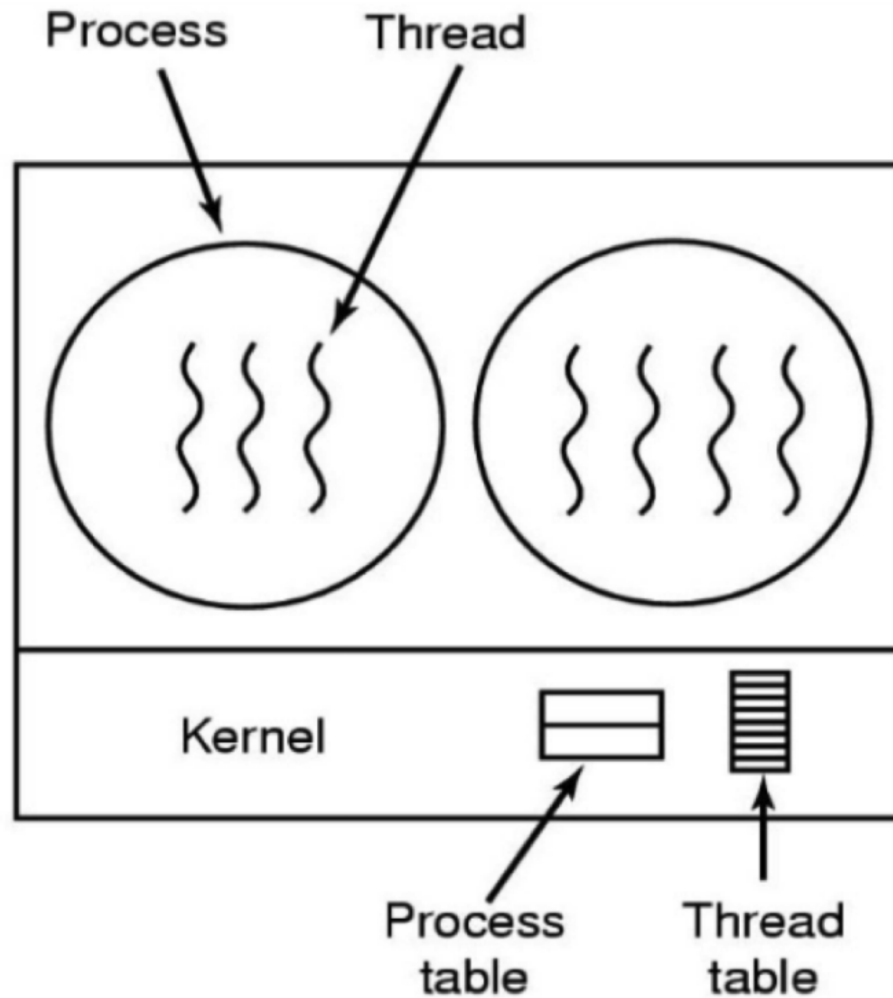
Pot fi planificate pe sisteme multiprocesor

Dezavantaje

Crearea si schimbarea de context este mai lenta

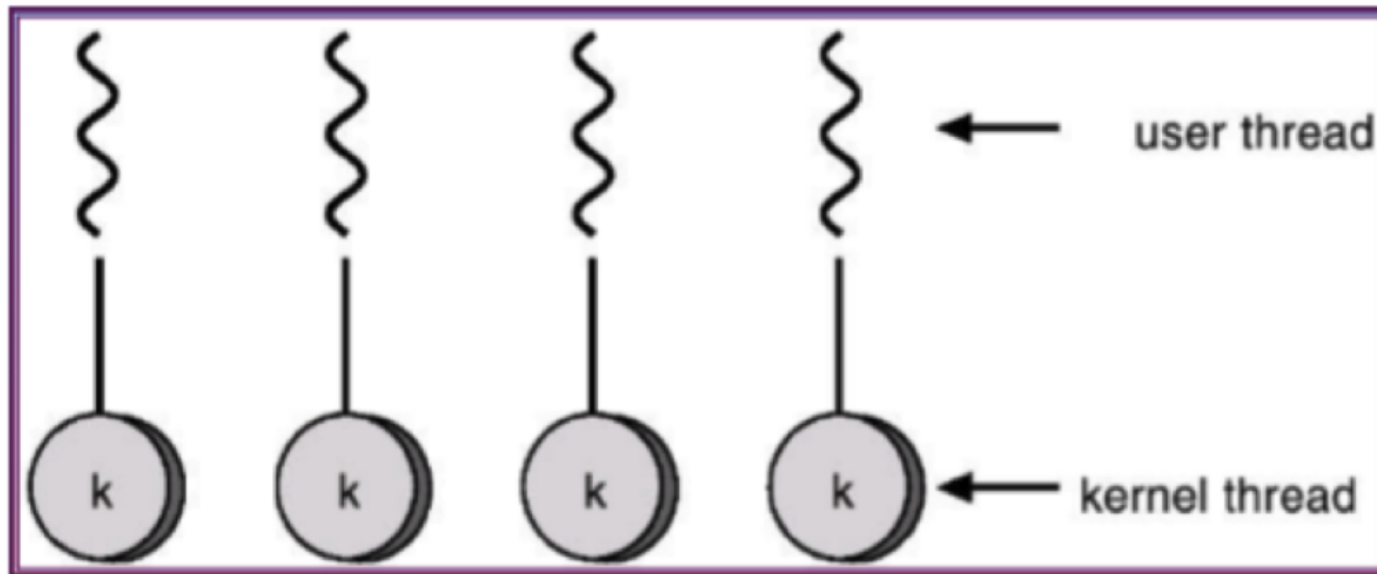
Prezi

Implementare thread-uri in kernel



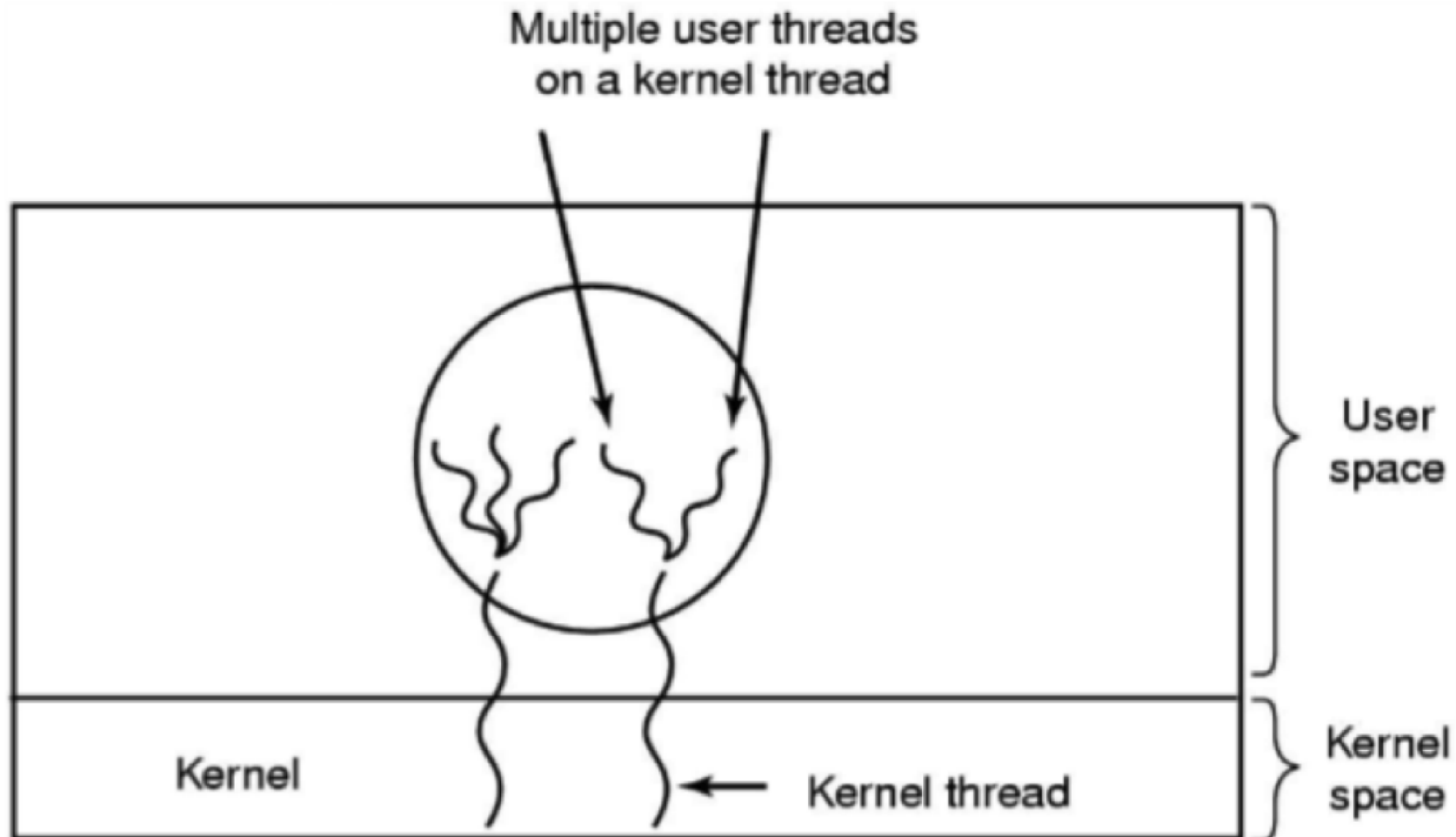
Prezi

Un kernel thread pentru fiecare thread utilizator



Prezi

Hibrid



Prezi

Sincronizare

Thread-urile ofera acces la date comune

Accesul trebuie mediat pentru a implementa programe corecte

Chiar si programele cu un singur thread pot crea probleme!

Dar cele cu mai multe thread-uri?

Exemplu

Reentranta

Reentranta in practica

Prezi

Exemplu

```
int total_bytes;
void update_statics(int j){
        total_bytes += j;
}

void signal_handler(){
        update_statistics(1);
}
```

Prezi

Reentranta

O functie este reentranta daca poate fi executata simultan de mai multe ori, fara a afecta rezultatul

Conditii necesare:

- nu lucreaza cu variabile globale/statice
- apeleaza doar functii reentrante

Reentranta este importanta (mai ales) in programe cu un singur thread din cauza semnalelor!

Prezi

Reentranta in practica

Multe functii de biblioteca seteaza variabila errno

Sunt acestea reentrante?

Depinde de implementare!

Anumite apeluri au versiuni reentrante: gethostbyname_r

Activare cu macroul _REENTRANT

Prezi

Depinde de implementare!

Anumite apeluri au versiuni reentrante: gethostbyname_r
Activare cu macroul _REENTRANT

Prezi

Thread Safety

O functie este thread-safe daca poate fi apelata din mai multe thread-uri in acelasi timp

Strategii implementare (cursul urmator)

- acces exclusiv
- semafoare
- monitoare
- thread-local storage
- reentranta
- operatii atomice

Prezi