UNIVERSITÀ DEGLI STUDI DI UDINE
Francesco Davini

Sistemi operativi

Introduzione

Crediti:

- IBML: 9CFU
- INF: 12CFU

Argomenti:

- Generalità sui sistemi operativi
- Processi e threat
- Scheduling delle CPU
- Sincronizzazione dei processi
- Blocco critico
- Gestione della memoria centrale
- Memoria secondaria
- Nozioni di base sul file system

Lab:

- La Bash di Unix
- Linguaggio C
- Introduzione alla programmazione di sistema UNIX/Linux Laboratorio extra: (solo per 12CFU) Programmazione di sistema di Linux/inux (penso che me le seguo lo stesso)

Il primo testo copre la teoria, il secondo compre laboratorio

Per programmare in Linux serve sapere C (so parte da zero)

Si trovano le cose principalmente su eLearning

Per fare l'esame:

- Lo scritto e orale sono su tutti gli argomenti sopracitati
- 1 prova scritta (fatta in laboratorio su moodle), una dozzina di domande relativamente semplici
- 1 orale (bisogna avere almeno la sufficienza per accederci)

Il corso è strutturato con: un terzo prima di Natale, due terzi dopo natale, si potrebbe valutare un parziale a metà gennaio, ma non è ancora sicuro

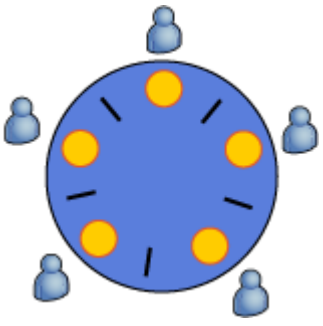
Soluzione con Priorità agli Scritti

```
semaphore W=1, R=1, mutex1=1, mutex2=1, mutex3=1;
int contR=0, contW=0;

void lettore(){
    wait (mutex3);
    wait (R);
    wait (mutex1);
    contR++;
    if (contR==1) wait (W);
    signal (mutex1);
    signal (R);
    signal (mutex3);
    // sezione critica di lettura
    wait (mutex1);
    if (contR==0) signal (W);
    signal (mutex1);
}

void scrittore(){
    wait (mutex2);
    contW++;
    if (contW==1) wait (R);
    signal (mutex2);
    wait (W);
    // sezione critica di scrittura
    signal (W);
    wait (mutex2);
    contW--;
    if (contW==0) signal (R);
    signal (mutex2);
}
```

I 5 filosofi Un tentativo di soluzione, utilizzando un vettore sem[N] di N semafori, uno associato per ogni bacchetta.



```
do {
    // pensare
    wait (forchetta[i]);
    wait (forchetta[(i+1)%N]);
    // mangiare
    signal (forchetta[i]);
    signal (forchetta[(i+1)%N]);
} while (true);
```

Questa soluzione però può portare a stallo (deadlock) se tutti i filosofi prendono contemporaneamente la forchetta di destra. Quindi nessuno mangia mai, da qui nasce il problema della **starvation** (fame).

Rimedi possibili:

- Aggiungere un controllo che la bacchetta di destra sia librera prima di prendere quella di sinistra (ma può portare a livelock)
- con lo stesso controllo ma introducendo un ritardo casuale prima di provare a prendere le bacchette
    - deallock evitato, la probabilità di starvation si riduce ma non si azzera
- introdurre un mutex che racchiuda la sezione critica della prima wait() alla ultima signal()
    - si evita deadlock e starvation, ma si perde parallelismo (un solo filosofo alla volta può mangiare)
- similmente al precedente: usando un semaforo si ammette che al massimo N-1 filosofi possano tentare di prendere le bacchette contemporaneamente
    - si evita deadlock e starvation, ma si perde parallelismo (al massimo N-1 filosofi possono mangiare contemporaneamente)
- soluzione "assimmetrica": un filosofo dispari prendere la prima bacchetta di sinistra, un filosofo pari prendere la prima bacchetta di destra
    - si evita deadlock e starvation, mantenendo il parallelismo

soluzuini senza assimetria:

```
(...)
RIGHT(i) = ((i+1) % N);
LEFT(i) = ((i+N-1) % N);
...stato[N]... // vettore di stati
semaphore mutex = 1;
semaphore Sem[N] = {0,0,0,...0}; // semafori uno per fiolosofo

void filosofo(int process){
    while (true){
        pensa();
```

```
        take_forks(process);
        mangia();
        put_forks(process);
    }
}

void take_forks(int process){
    wait(mutex);
    stato[process] = HUNGRY;
    test(process);
    signal(mutex);
    wait(Sem[process]);
}

void put_forks(int i){
    wait(mutex);
    stato[i] = THINKING;
    test(LEFT(i));
    test(RIGHT(i));
    signal(mutex);
}

void test(int i){
    if (stato[i] == HUNGRY &&
        stato[LEFT(i)] != EATING &&
        stato[RIGHT(i)] != EATING){
            stato[i] = EATING;
            signal(Sem[i]);
    }
}
```