# Esame 8/2/2023

## Esercizio 1

Una chiamata di sistema viene eseguita da un processo per richiedere servizi dal sistema operativo. Le chiamate di sistema vengono implementate tramite trap; una trap è un interrupt software provocato dall'esecuzione di una determinata istruzione del programma in esecuzione. Per effetto della trap, la CPU passa da user mode a kernel mode, dove il trap handler gestisce la chiamata. Completata l'operazione, il controllo viene ritornato al processo passando in user mode.

Per quanto riguarda le gestione dei processi, il sistema operativo consuma tempo per svolgere i context switch (passaggio di esecuzione da un processo ad un altro), che quindi non devono essere troppo frequenti, e consuma spazio nella gestione della tabella dei processi e delle pagine, che non deve essere quindi allocata per tutto lo spazio di indirizzi poteziale del processo. Per quanto riguarda la gestione della mamoria, il SO consuma tempo nella traduzione degli indirizzi, specialmente con la paginazione, che viene velocizzata con il TLB; nella paginazione su richiesta, in quanto pagine non presenti in memoria (page fault) vanno caricate, e il tempo d'accesso medio alla memoria rischia di salire molto se i page fault sono frequenti (vanno ottimizzati quindi tramite algoritmi di rimpiazzamento ed eventualmente facendo swap out dei processi per evitare thrashing).

## Esercizio 2

Test and Set Lock è un'istruzione di livello ISA (TSL RX,LOCK). Legge i contenuti della parola LOCK nel registro RX e memorizza un valore non nullo all'indirizzo di memoria LOCK. L'operazione di lettura e memorizzazione sono indivisibili, poichè la CPU blocca il bus di memoria. Quando LOCK è 0, qualsiasi processo può impostarlo ad 1 usando l'istruzione TSL e procendendo con la lettura o scrittura della memoria condivisa; alla fine dell'operazione, il processo imposta LOCK a 0.

```
enter_region:
    TSL REGISTER,LOCK   //copia LOCK in registro e impostalo a 1
    CMP REGISTER,#0     //controlla se lock = 0
    JNE enter_region    //se != 0, loop
    RET                 //ritorna al chiamante, entra in regione critica

leave_region:
    MOVE LOCK,#0        //memorizza 0 nel LOCK
    RET                 //return
```

In linguaggio di alto livello:

```
Inizializzazione:
lock = 0;

Per entrare in sezione critica:
while(TestAndSet(&lock));

In uscita dalla sezione critica:
lock = 0;
```

Questa soluzione non soddisfa il requisito di attesa limitata.

## Esercizio 5

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <fcntl.h>

int main(int argc, char **argv) {
    if(argc != 3) {
        printf("Utilizzo: ./shell <numero processi da eseguire> <nome file>
");
        exit(1);
    }
    if(argv[1] <= 0) {
        printf("Inserire intero maggiore di 0");
        exit(1);
    }

    int n = atoi(argv[1]);
    int f = creat(argv[2], S_IRUSR | S_IWUSR);
    char s[64];
    int status;
    pid_t pid;

    printf("> ");
    while(fgets(s, sizeof s, stdin) != NULL) {
        if(s[strlen(s)-1] == '\n') {
            s[strlen(s)-1] = 0;
        }
        if(strcmp(s, "exit") == 0) {
            close(f);
            exit(0);
        }
        for(int i = 0; i < n; i++) {
            pid = fork();
            if(pid == 0) {
                dup2(f, 1);
                execlp(s, s, (char *)0);
                exit(0);
            }
            if(n % 2 == 0) {
                waitpid(pid, &status, 0);
            }
        }
        printf("> ");
    }
}
```

## Esercizio 6

Le funzioni sviluppate non sono corrette in quanto manca un semaforo mutex per dare ad un processo accesso esclusivo alle variabili di stato.

```
void take_forks(int i) {
    down(&mutex);
    state[i] = HUNGRY;
    test(i);
    up(&mutex);
    down(&s[i]);
}

void put_forks(int i) {
    down(&mutex);
    state[i] = THINKING;
    test(LEFT);
    test(RIGHT);
    up(&mutex);
}

void test(int i) {
    if(state[i] == HUNGRY && state[LEFT] != EATING && state[RIGHT] !=
EATING) {
        state[i] = EATING;
        up(&s[i]);
    }
}
```