Selezione di esercizi di sistemi operativi (parte 1)

1 – Cos'è un sistema operativo? Quali sono le principali funzioni e gli obiettivi di un sistema operativo? Presentare qualche esempio di come sono raggiunti tali obiettivi

Traccia di soluzione

Un sistema operativo è un programma che controlla l'esecuzione dei programmi applicativi e funge da interfaccia tra l'utente di un computer e l'hardware.

Funzioni:

-Alloca risorse, evitando conflitti tra i vari processi che le utilizzano
-Controlla il flusso di esecuzione dei programmi, gestendo le situazioni di errore

Obiettivi:

-Eseguire in maniera opportuna i programmi → es. protezione della memoria
-Facilitare l'uso del computer → es. system call
-Far sì che le risorse del computer siano usate in modo efficiente → es. Multitasking: più programmi in esecuzione

2 – Spiegare i meccanismi di accesso ai dispositivi di I/O con polling e attraverso interrupt Quali sono le limitazioni del polling? Come vengono gestiti gli interrupt dal sistema operativo?

Traccia di soluzione

Polling -> Verifica ciclica dello stato del dispositivo attraverso i suoi registri per rilevare se si è verificato un evento.
Interrupt -> Al verificarsi di un evento Il dispositiva invia alla CPU un segnale interrompendo il programma in esecuzione. Il controllo viene passato ad una funzione di sistema operativo (ISR) che implementa il protocollo di I/O di quello specifico device. L'indirizzo della ISR associata all'interrupt si trova nll'interrupt vector. Al termine dell'ISR, il sistema operativo metterà in esecuzione un nuovo processo.

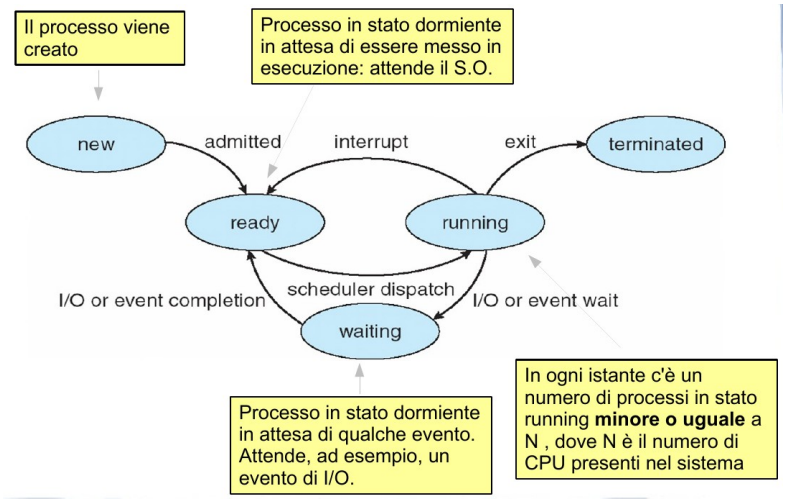
Limitazioni polling: tempo perso su cicli inutili.

Nella politica ad interrupt, il sistema operativo:
-Completa il context switch iniziato dalla CPU
-Riconosce il dispositivo che ha provocato l'interruzione
-Gestisce l'interruzione eseguendo l'ISR
-Al termine, schedula il nuovo processo da eseguire e resetta il mode bit in user mode prima di eseguire il dispatching

3 - Cos'è un processo? Quali sono gli elementi che definiscono un processo? Fornire una breve descrizione di ognuno di essi. Quali sono i possibili stati in cui si può trovare un processo e quali sono gli eventi che causano il passaggio da uno stato all'altro? (Fornire un possibile diagramma degli stati)

Traccia di soluzione

Processo: Entità attiva che rappresenta il programma in esecuzione, gestito dal sistema operativo per mezzo di una struttura dedicata (PCB)
Elementi che definiscono un processo:
-Un process ID (PID)
-Il contenuto dei registri di CPU (PC, registri di ALU, ...)
-Una porzione di memoria ad esso dedicata ("spazio di indirizzamento), che include:
    -Il programma o “eseguibile” (nella text section)
    -Le variabili globali (nella data section)
    -Lo stack
    -L'heap
-Una lista di file gestiti dal processo



4 – Che cos'è lo scheduler? Descrivere le differenze tra long, mid e short term scheduler. Quali possono essere gli obiettivi di uno short term scheduler? Con che strutture vengono gestiti dallo scheduler i processi in esecuzione?

Traccia di soluzione

Scheduler → sceglie i processo da eseguire o portare in RAM tra quelli pronti per essere eseguiti

-Long-term scheduler: seleziona quali dei processi che possono essere eseguiti immediatamente devono essere spostati in RAM, inserendoli nella ready queue

-Medium-term scheduler: rimuove dalla memoria principale processi in esecuzione ma inattivi, viceversa riporta in memoria principale processi che tornano nella ready queue
-Short-term scheduler (o CPU scheduler): seleziona il prossimo processo da eseguire

Obiettivi di uno short-term scheduler :

    -massimizzare l'uso delle risorse
    -minimizzare i tempi di reazione

Lo scheduler usa delle code (queue) per gestire i processi in esecuzione, di solito implementate come linked list (o doubly linked list) di PCB:

-Job queue: tutti i processi in esecuzione
-Ready queue: processi pronti per essere eseguiti
-Device queue: l'insieme dei processi in attesa di qualche operazione di I/O

5 - Descrivere brevemente le funzioni UNIX fork(), exec() e wait().
Cosa stampa il seguente programma? Perché? In che sezione di memoria (data, stack o heap) vengono memorizzate le variabili local_val, global_val e l'area di memoria puntata da local_val?

```
int global_val;

int main()
{
  pid_t pid;
  int *local_val = (int *)malloc(sizeof(int));
  global_val = 0;
  *local_val = 1;

  pid = fork();
  if(pid == 0)
  {
    global_val = 100;
    (*local_val)++;
    printf("CHILD : global_val = %d local_val = %d\n", global_val, *local_val);

    return 0;
  }
  else
  {
    wait(NULL);
    printf("PARENT : global_val = %d local_val = %d\n", global_val, *local_val);

    free(local_val);
    return 0;
  }
}
```

Traccia di soluzione

-fork(): crea un processo figlio che è una copia del processo padre con PID diverso.
-exec(): carica nel segmento text della memoria del processo chiamante un nuovo programma (il contenuto precedente viene distrutto) ed Inizia ad eseguire il nuovo programma
-wait(): attende la terminazione di un processo figlio precedentemente creato con fork(), ritorna il valore ritornato dal processo figlio.

Il programma stampa a video:

CHILD : global_val = 100 local_val = 2
PARENT : global_val = 0 local_val = 1

Spiegazione: La memoria assegnata al processo figlio è un duplicato della memoria assegnata al processo padre.

6 - Che differenza c'è fra un processo e un thread? Cosa accade quando il processo padre che ha creato un thread termina? La priorità di schedulazione di un thread è influenzata dalla priorità di schedulazione del processo che l'ha creato?

Traccia di soluzione

Un processo è un'entità a sé stante, con un proprio spazio di indirizzamento indipendente dallo spazio di indirizzamento del processo padre che l'ha creato. Un thread può essere definito come un "lightweight process" creato da un processo con il quale condivide parte dello spazio di memoria (dati globali e heap) mentre program counter, valori nei registri e stack sono indipendenti. La creazione e il context switch tra thread sono molto più rapide rispetto alla creazione e il context switch tra processi convenzionali. Quando il processo padre termina,
termineranno tutti i suoi thread. I thread possono avere priorità di schedulazione differente rispetto al processo padre.

7 – Descrivere la differenza tra uno scheduler non-preemptive e preemptive. Evidenziare le possibili problematiche che possono generare questi due tipi di scheduler. Gli algoritmi First-Come, First-Served, Shortest-Job-First e Round-Robin a quale tra queste due categoria appartengono?

Traccia di soluzione

-Non-preemptive: il processo in esecuzione rimane tale finché non lascia esplicitamente il controllo della CPU ed entra in una waiting queue (a seguito di una richiesta di I/O) oppure termina.
Problematiche: se un processo in esecuzione non accede ad alcuna risorsa di I/O e non termina, nessun altro processo verrà mai eseguito -> starvation.

-Preemptive: a seguito di una qualsiasi interruzione, lo scheduler può decidere se proseguire nell'esecuzione del processo corrente oppure schedulare l'esecuzione di un nuovo processo.
Problematiche: (a) E' necessario fornire degli strumenti di sincronizzazione tra processi (es. i semafori) (b) Interruzione su interruzione

L'intervento dello scheduler può essere richiesto quando un processo cambia di stato (o queue):

1) Running state → waiting state (es. syscall per accesso I/O)
2) Running state → ready state (es. interrupt “esterno”, dovuto a I/O, timer, ...)
3) Waiting state → ready state (es. l'operazione di I/O attesa dal processo è completata)
4) Running state → processo terminato

1,4 -> non-premptive
1,2,3,4 -> preemptive

-First-Come, First-Served: non-preemptive
-Shortest-Job-First : entrambi(Non-preemtive: si attende ogni volta la terminazione del processo attualmente in esecuzione, anche se nel frattempo è arrivato in ready queue un processo con minore durata del prossimo CPU-burst. Preemtive: viceversa, anche chiamato Shortest-remaining-time-first)
-Round-Robin: preemptive

8 - Dati 5 processi con i seguenti tempi di arrivo in ready queue e CPU burst:

Processo	**Tempi di arrivo**	**Tempi di burst**
P1	2	20
P2	5	7
P3	10	13
P4	16	7
P5	21	7

Calcolare average waiting time e tempo medio di turnaround in caso di round-robing scheduling con quanto di 6 unità di tempo.

Traccia di soluzione

AWT: (32+15+33+29+27)/5 = 136/5 = 27.2
Tempo medio di turnaround: ((54-2)+(27-5)+(56-10)+(52-16)+(55-21))/5 = 190/5=38

9 - Definire il problema di race condition, fornendo un frammento di pseudo-codice di esempio che presenta tale problema, descrivendo la temporizzazione degli eventi che portano al problema. Modificare quindi l'esempio per risolvere il problema.

Traccia di soluzione

Race condition è un problema che può presentarsi in caso di accesso concorrente di più processi ad una risorsa condivisa (es. memoria o file) per cui il risultato finale dell'esecuzione dei processi dipende dalla sequenza e dalla schedulazione con cui vengono eseguiti. In altre parole, fissato lo stato iniziale (dati), lo stesso codice può portare a risultati differenti in caso di sequenze di schedulazione differente.

Esempio:

Sia data la variabile x, inizializzata a 0, condivisa tra due processi A e B che stanno eseguendo lo stesso codice:

```
for ( int i = 0; i <  1000; i++ )
{
        x = x + 1;
}
```

Se i processi sono eseguiti in sequenza, il valore finale dovrebbe essere 2000. Se i processi sono eseguiti in parallelo, il valore finale potrebbe essere < 2000.

L'istruzione x = x + 1 viene divisa in 3 istruzioni:

Load X
Aggiungi 1
Store X

Se uno dei due processi viene de-schedulato dopo il load e viene schedulato l'altro, entrambi effettueranno il load memorizzando lo stesso valore.
Soluzione: mutex lock per la regione critica x = x + 1.

10 – Descrivere il problema dell'inversione delle priorità in caso di accesso concorrente ad una risorsa sincronizzato con un mutex lock. Fornire una possibile soluzione

Traccia di soluzione

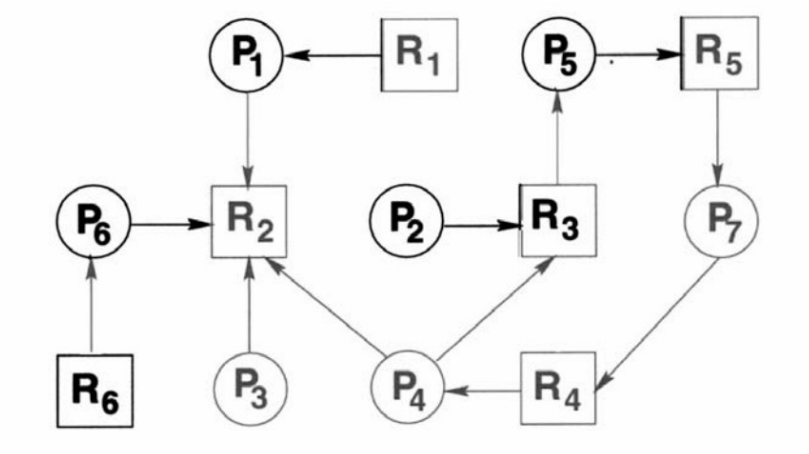
Un processo a bassa priorità A sta eseguendo una critical section su una risorsa condivisa con un processo ad alta priorità C. Durante l'esecuzione, il processo ad alta priorità chiede l'accesso alla risorsa condivisa: essendo temporaneamente occupata da A, C viene messo in attesa su una coda associata con la mutex lock. Se in questo frangente, un processo B con priorità compresa tra A e C entra nella ready queue, A verrà (preemtive) tolto dall'esecuzione a favore di B. C dovrà attendere anche B, nonostante la sua priorità sia più alta.
Possibile soluzione: quando C richiede la risorsa, A eredita la sua priorità, che poi verrà ristabilita al valore iniziale.

11 - Un sistema è composto da 7 processi, P1, ..., P 7 , e da 6 risorse distinte, ognuna delle quali possiede una sola istanza, condivise, R1, ..., R 6

-P1 occupa R1 e richiede R 2;
- P 2 non occupa risorse e richiede R 3;
-P 3 non occupa risorse e richiede R 2;
-P 4 occupa R 4 e richiede sia R 2 sia R 3;
-P 5 occupa R 3 e richiede R 5;
-P 6 occupa Re e richiede R 2;
-P 7 occupa R5 e richiede R 4.

Il sistema si trova in stato di deadlock?

Traccia di soluzione



RAG con loop, risorse singole -> deadlock

12 - Dati 4 processi, e 3 tipi di risorse:

Processi	**Allocation**			**Max**			**Totale risorse**		
	A	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
P1	0	1	4	4	1	4	5	8	16
P2	2	0	1	3	1	4			
P3	1	2	1	5	7	13			
P4	1	0	3	1	1	6			

Utilizzando l'algoritmo del banchiere, si determini:

1) Se il sistema è in uno stato sicuro;
2) Se l'assegnazione di 1 istanza della risorsa A a P1 garantisce il mantenimento dello stato sicuro;
3) Se l'assegnazione di 6 istanze di C a P 3 garantisce il mantenimento dello stato sicuro.

Traccia di soluzione

1) Calcolo matrice need e available:

Matrice need:

A	**B**	**C**
4	0	0
1	1	3
4	5	12
0	1	3

Matrice available:

A	B	C
1	5	7

Sistema in stato sicuro, sequenza <p2, p4, p1, p3>

2) P1 richiede un'istanza di tipo A, richiesta possibile (< need, = available)

Matrice available:

A	B	C
0	5	7

Matrice allocation:

A	B	C
1	1	4
2	0	1
1	2	1
1	0	3

Matrice need:

A	B	C
3	0	0
1	1	3
4	5	12
0	1	3

Sistema in stato sicuro, sequenza <P4, P2, P1, P3>

3) P3 richiede 6 istanze di tipo C

Matrice available:

A	B	C
1	5	1

Matrice need:

A	B	C
4	0	0
1	1	3
4	5	6
0	1	3

Nessun processo per cui need <= available, stato non sicuro

13 – Si consideri un sistema di allocazione della mamoria basato sulla paginazione. Lo spazio logico indirizzabile sia al massimo di 64 byte, suddiviso in pagine da 4 byte.

La memoria fisica sia costituita da 256 byte.

-Da quanti bit sono costituiti gli indirizzi logici e gli indirizzi fisici?
-Da quanti bit sono costituiti i numeri di pagina?
-Da quanti bit sono costituiti i numeri di frame?
-Ad un dato istante, la tabella delle pagine (page table) di un processo è la seguente:

Numero pagina	Numero frame
0	12
1	1
2	17
3	62
4	11
5	16
6	61
7	12

Tradurre in indirizzi fisici i seguenti indirizzi logici: 0, 2, 9, 11, 32, 30, 26

Traccia di soluzione

Indirizzi logici 6 bit, fisici 8 bit
Numero di pagina: 4 bit
Numero di frame: 6 bit

0 -> 4 * 12 = 48
2 -> 50
9 -> 69
11 -> 71
32 -> pagina 8, errore
30 ->50
26 -> 246

14 - Cos'è il binding degli indirizzi e quali sono le motivazione che ne hanno portato l'introduzione? Spiegare in che fasi può essere effettuato il binding

Traccia di soluzione

Il binding è l'associazione di istruzioni e dati a indirizzi di memoria fisica.
Motivazione: Negli algoritmi di allocazione, ai processi possono essere spesso associati spazi di memoria diversi, ovvero ad un programma quasi mai vengono assegnati gli stessi indirizzi di memoria ad ogni esecuzione per .

Possibili fasi di binding:
-Compilazione: se la posizione in memoria del processo è nota a priori, può essere generato un codice assoluto; se la locazione iniziale cambia, è necessario ricompilare il codice;
-Caricamento: se la posizione in memoria non è nota in fase di compilazione, è necessario generare codice rilocabile;

-Esecuzione: se il processo può essere spostato a run–time da una zona di memoria ad un'altra, il binding viene rimandato fino al momento dell’esecuzione. È necessario un opportuno supporto hardware per mappare gli indirizzi (ad esempio attraverso registri base e limite).