Sistemi Operativi
10 luglio 2007
Compito scritto

Si risponda ai seguenti quesiti, giustificando le risposte.

1. Si riassuma la storia dei sistemi operativi, descrivendone le caratteristiche importanti introdotte nel corso del tempo.

   Risposta: I primi calcolatori (grosse e rudimentali macchine basate su valvole ed interruttori) erano prive di sistemi operativi e tutta la gestione del sistema era a carico di un gruppo di programmatori che operavano in puro linguaggio macchina, agendo direttamente sui vari interruttori. Con l'avvento dei transistor, vennero introdotti i primi sistemi batch con i primi sistemi operativi monoutente e monoprogrammati: tipicamente veniva eseguito un job di seguito all'altro tramite la lettura di apposite schede perforate (tutta la memoria era a disposizione del programma da eseguire, eccezion fatta per la porzione necessaria ad ospitare un rudimentale sistema operativo). Successivamente con la comparsa di nuove periferiche, per sfruttare la CPU anche durante le operazioni di I/O di un processo, si è introdotto il concetto di multiprogrammazione in cui più processi vengono caricati in memoria e la CPU, durante l'attesa per un'operazione di I/O da parte del processo in corso d'esecuzione, viene assegnata ad un altro processo. In questo modo si tende a sfruttare maggiormente una risorsa preziosa come il microprocessore (risorsa che precedentemente rimaneva inutilizzata durante le operazioni di I/O).

   Successivamente l'idea alla base della multiprogrammazione è stata estesa portando al *time-sharing*, ovvero, una variante in cui il/i processore/i esegue/ono più processi commutando le loro esecuzioni con una frequenza sufficientemente elevata da permettere a ciascun utente di interagire con il proprio programma durante la sua esecuzione. In questo modo ogni processo del sistema si vede assegnare la CPU per un certo intervallo di tempo, rendendo possibile l'utilizzo contemporaneo del sistema da parte di più utenti (multiutenza).

   Con il miglioramento della tecnologia e l'aumento dell'integrazione dei circuiti divenne presto possibile produrre dei computer a basso costo per uso domestico e nacquero i primi sistemi operativi per microcomputer, prendendo spunto per molti versi dalla tecnologia ormai consolidata dei sistemi operativi per mainframe e minicomputer.

2. (a) Si confrontino le implementazioni dei thread a livello kernel e a livello utente.
   (b) Si diano esempi di situazioni in cui è vantaggioso il primo o il secondo tipo di implementazione.

   Risposta:

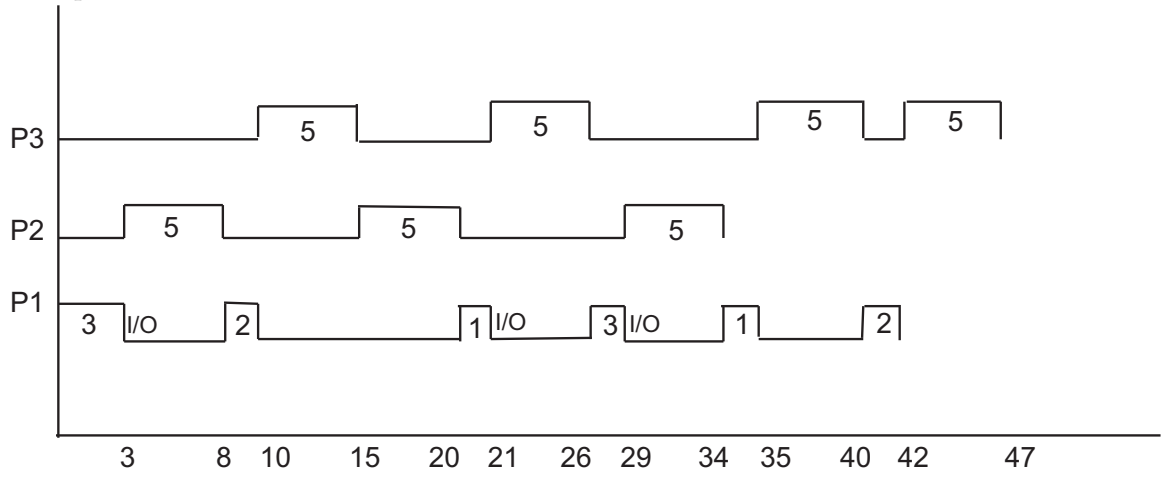
   (a) I thread sono unità di esecuzione all'interno di uno stesso processo; ogni thread è caratterizzato da un proprio PC, da un proprio insieme dei valori dei registri, da un proprio stack ed un proprio stato di esecuzione, mentre tutte le risorse rimanenti (spazio in memoria, file aperti ecc.) vengono condivise fra thread appartenenti allo stesso processo. Vi sono due possibili implementazioni dei thread: a livello utente ed a livello kernel. Nel primo caso ogni processo mantiene una tabella dei thread propria in cui viene tenuta traccia di tutti i thread del processo (PC, valori dei registri, stack, stato di esecuzione). Nel caso invece di un'implementazione a livello kernel viene tenuta, a fianco della tabella dei processi, una tabella dei thread nello spazio di memoria del nucleo del sistema operativo (quindi c'è un'unica tabella per tutti i thread in esecuzione nel sistema). Esistono anche delle implementazioni *ibride* in cui uno o più thread a livello utente (a seconda del grado di parallelismo che si intende ottenere) vengono mappati su un thread a livello kernel.

   (b) I thread a livello utente sono convenienti per processi CPU-bound, ovvero, per processi che non necessitino di unattività di I/O intensiva (che bloccherebbe il processo con tutti i suoi thread) in cui sia possibile scomporre e parallelizzare il lavoro. Infatti con i thread a livello utente tutte le operazioni di manipolazione dei thread (creazione, cancellazione, context switch ecc.) sono gestite tramite librerie in spazio utente che non necessitano quindi di trap al kernel e che risultano molto veloci ed efficienti. Ad esempio in un word processor è utile far eseguire il task di correzione ortografica in un thread a livello utente separato, rispetto al thread che si occupa di effettuare il rendering a video del testo. Invece nel caso di processi che eseguano molto I/O (ad esempio web server) conviene utilizzare dei thread a livello kernel, in quanto il blocco di un singolo thread a livello utente, per l'attesa del compimento di un'operazione di I/O, comporta generalmente il blocco dell'intero processo (ovvero, di tutti i suoi thread).

Sistemi Operativi
10 luglio 2007
Compito scritto

3. I processi P_1, P_2, P_3 sono in coda ready nell'ordine specificato all'istante 0. P_1 è un processo ciclico che alterna 3ms di esecuzione a 1ms di operazioni di I/O. Il tempo totale di esecuzione per P_1 è pari a 12ms. I processi P_2 e P_3 hanno una richiesta di CPU pari a 15ms e 20ms, rispettivamente e non effettuano I/O. Si calcolino i tempi di attesa per i 3 processi, sapendo che l'algoritmo di scheduling della CPU è RR con q=5ms e che un processo, che rilascia la CPU prima dello scadere del suo quanto, torna all'inizio della coda ready con il resto del quanto.

   Risposta:



   Tempi di attesa: $T_1 = 4+10+4+4+5 = 27\ ms$, $T_2 = 3+7+9 = 19\ ms$, $T_3 = 10+6+9+2 = 27\ ms$.

4. (a) Si descriva cosa si intende per interruzione precisa.
   (b) Elencare i problemi che derivano dall'avere delle interruzioni imprecise.

   Risposta:

   (a) Un'interruzione si dice precisa quando gode delle seguenti quattro proprietà:
      1. il program counter viene salvato in un posto noto,
      2. tutte le istruzioni che precedono quella puntata dal program counter sono state completamente eseguite,
      3. nessuna delle istruzioni che seguono quella puntata dal program counter è stata eseguita,
      4. lo stato di esecuzione dell'istruzione puntata dal program counter è noto.

   (b) Nel caso in cui si abbiano interruzioni imprecise è difficile riprendere l'esecuzione in modo esatto in hardware: la CPU si limita a "riversare" sullo stack tutta l'informazione relativa allo stato corrente, lasciando che sia il sistema operativo a capire che cosa debba essere fatto. In questo modo rallentano le fasi di ricezione degli interrupt e di context-switch/ripristino dell'esecuzione, provocando grosse latenze.

5. Uno dei problemi con una struttura di directory a grafo è la presenza di cicli da evitare durante la visita del filesystem e la creazione di garbage in seguito ad operazioni di cancellazione. Si illustrino le due soluzioni adottate dai sistemi operativi della famiglia Unix per ovviare a tali problemi (sugg.: pensare ai limiti imposti ai link hard e simbolici).

   Risposta: le due soluzioni adottate dai sistemi operativi della famiglia Unix per ovviare a tali problemi sono:

   1. permettere la creazione di link ai soli file (soluzione adottata per i link hard),
   2. limitare il numero di link attraversabili (soluzione adottata per i link simbolici).

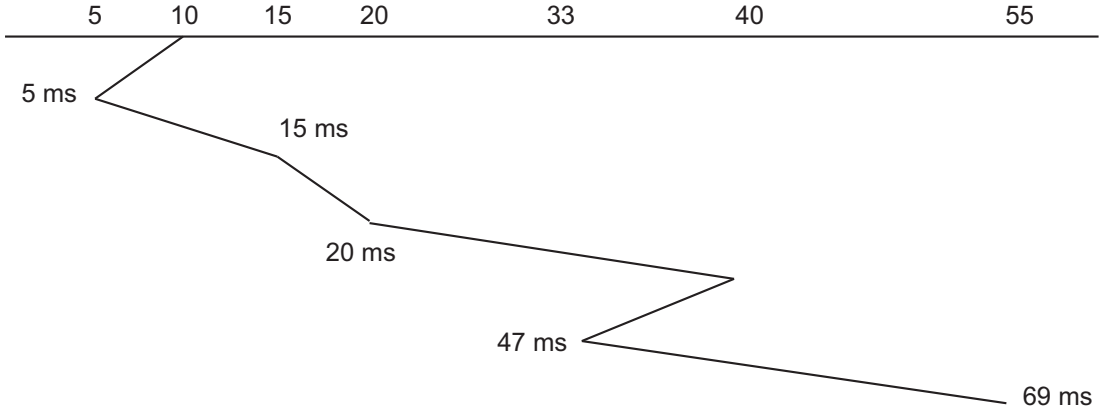
6. Si consideri un disco gestito con politica SSTF. Inizialmente la testina è posizionata sul cilindro 10; lo spostamento ad una traccia adiacente richiede 1 ms. Al driver di tale disco arrivano richieste per i cilindri 5, 20, 15, 55, 33, rispettivamente agli istanti 0 ms, 2ms, 5 ms, 19 ms, 40 ms. Si trascuri il tempo di latenza.

   1. In quale ordine vengono servite le richieste?

2. Il tempo di attesa di una richiesta è il tempo che intercorre dal momento in cui è sottoposta al driver a quando viene effettivamente servita. Qual è il tempo di attesa medio per le cinque richieste in oggetto?

Risposta:

1. Le richieste vengono soddisfatte nell'ordine: 5, 15, 20, 33, 55, come risulta dal seguente diagramma:



2. Il tempo di attesa medio per le cinque richieste in oggetto è $\frac{(5-0)+(15-5)+(20-2)+(47-40)+(69-19)}{5} = \frac{5+10+18+7+50}{5} = \frac{90}{5} = 18\ ms$.

7. Descrivere brevemente come avviene un attacco basato sulla tecnica del buffer overflow.

Risposta: Siccome la maggior parte dei sistemi operativi e dei programmi di sistema sono scritti in C (per ragioni di efficienza) e quest'ultimo non mette a disposizione dei meccanismi per impedire che si facciano dei riferimenti oltre i limiti dei vettori, è molto facile (ad esempio, passando in input al programma una stringa di caratteri sufficientemente lunga) andare a sovrascrivere in modo improprio delle zone di memoria con effetti potenzialmente disastrosi. La tecnica del buffer overflow sfrutta proprio questa vulnerabilità; infatti, se il vettore è locale ad una funzione C, allora la memoria ad esso riservata verrà allocata sullo stack e sarà possibile, con un indice che ecceda il limite massimo consentito per il vettore, andare a modificare direttamente lactivation record della funzione. In particolare, modificando lindirizzo di ritorno, è possibile di fatto *redirezionare* lesecuzione del programma, provocando lesecuzione di codice potenzialmente pericoloso. La tecnica del buffer overflow viene in particolare usata per eseguire uno ShellCode, ovvero, un programma in linguaggio assembly che esegue una shell (e.g., /bin/sh in Unix o command.com in Windows) da cui l'attaccante può iniziare comodamente altre azioni pericolose nei confronti del sistema. In questo modo quando la funzione prova a restituire il controllo al chiamante, leggendo sullo stack lindirizzo di ritorno, in realtà provoca lesecuzione dello ShellCode.

Oltre ad *iniettare* lindirizzo di inizio dello ShellCode sullo stack, bisogna riuscire anche ad inserire il codice stesso dello ShellCode nello spazio di memoria indirizzabile dal processo vulnerabile. A tale scopo, una tecnica relativamente semplice consiste nel definire una variabile dambiente che lo contenga come valore e richiamare il programma vulnerabile passandogli tale ambiente.

Il punteggio attribuito ai quesiti è il seguente: 4, 5, 6, 4, 3, 4, 5 (totale: 31).