Università degli Studi di Messina – A.A. 2014/2015

Progetto Sistemi Operativi

Simulazione allocazione della memoria

Salvatore Bertoncini, matr. 445857
16/07/2015

Sommario

1 Richiesta 2
2 Algoritmi di allocazione 2
3 Implementazione 3
3.1 Ambiente e Tool di sviluppo 3
3.2 Architettura applicazione 3
3.3 Modalità d'uso 4
4 Statistiche 4
5 Diagrammi UML 5
5.1 Class Diagram 5
6 Manuale d'uso 5
7 Risultati 6

1 Richiesta

Viene richiesta l'implementazione di un applicativo che simuli tre delle tecniche usate per eseguire l'allocazione della memoria utilizzando una free-list: first-fit, best-fit e worst-fit. L'applicativo deve prevedere una situazione iniziale (randomica) della memoria e confrontare il risultato finale ottenuto dalle tre tecniche. Supponendo che ogni confronto costi una unità, si analizzino le differenze in termini di costo e in termini di frammentazione esterna. L'applicativo deve, inoltre, prevedere i boundary tag per facilitare l'unione delle aree di memoria libere. Il parametro "Numero di simulazioni" deve essere inserito da linea di comando. L'applicativo deve effettuare delle simulazioni al fine di esaminare il diverso comportamento delle tre tecniche mostrando ad ogni simulazione lo stato del sistema sia a video che su file di log.

2 Algoritmi di allocazione

Gli algoritmi di allocazione utilizzati per il progetto sono i seguenti:

- **First Fit**: è la tecnica più semplice e prevede l'occupazione della prima partizione libera e grande abbastanza da contenere il processo che si incontra durante la scansione della lista delle partizioni libere.
- **Best Fit**: prevede l'allocazione del processo nella partizione la cui dimensione è al contempo la più piccola possibile tra tutte quelle in lista, ma sufficientemente grande per poter ospitare il processo
- **Worst Fit**: segue l'idea esattamente opposta rispetto alla Best Fit, alloca cioè il processo nella partizione più grande disponibile

3 Implementazione

3.1 Ambiente e Tool di sviluppo

L'applicazione è stata realizzata utilizzando il linguaggio di programmazione **Python 2.7**, in ambiente **Manjaro Linux x64**, derivata del famoso **Arch Linux**. Come editor di testo si è scelto di adottare **Sublime Text 2**. Il documento corrente è stato scritto in Microsoft Word 2007, in ambiente Windows 7.

3.2 Architettura applicazione

L'architettura del seguente applicativo Python è stata realizzata seguendo i canoni della programmazione OOP (Object-Oriented Programming), cercando di massimizzare l'usabilità delle classi.

Queste frammenti di codice rappresentano l'interfaccia di alcune delle classi più importanti:

- **Blocco di memoria**: classe Python Blocco, che rappresenta la porzione di area di memoria presente nell'allocatore. Ha un byte di inizio e un byte di fine, e può essere occupato o libero.

```
11  #classe che rappresenta ogni singolo blocco (partizione) della memoria
12  class Blocco:
13      #COSTRUTTORE
14      def __init__(self, inizio, fine):
15          self.inizio = inizio #rappresenta l'indice di inizio memoria occupata (il "byte" di memoria iniziale)
16          self.fine = fine #rappresenta l'indice di fine memoria occupata (il "byte" di memoria finale)
17          self.occupato = False #indica se il blocco di memoria e' stato assegnato a un processo (True) o e' libero (False, come da default)
18
19      #GETTERS
20      def dimensioneBlocco(self): #restituisce la dimensione del blocco semplicemente sottraendo fine ad inizio
21          return (self.fine - self.inizio)
22
```

- **Memoria**: classe Python Memoria, che rappresenta l'intera memoria, la quale è composta da una lista di blocchi di memoria di tipo Blocco. La memoria ha una dimensione.

```
32  #classe che rappresenta l'intera memoria
33  class Memoria:
34      #COSTRUTTORE
35      def __init__(self, dimensione):
36          self.dimensione = dimensione #rappresenta la dimensione totale della memoria, di default 1024
37          self.reset ()
38
39      def reset(self):
40          self.blocchiMemoria = [] #rappresenta la lista di blocchi (partizioni) della memoria
41
42          self.blocchiMemoria.append(Blocco(0,self.dimensione)) #crea il primo blocco unico di memoria
43
44          self.freeList = []
45
46          self.freeList.append(Blocco(0,self.dimensione))
47
```

- **Processo**: classe Python Processo, che rappresenta il processo del nostro sistema, il quale ha una sua dimensione, ed è composto da un tempo di ingresso in esecuzione, che andrà anche a determinare quale dei processi verrà eseguito prima degli altri, un tempo di esecuzione, che determinerà per quanti cicli di clock il processo dovrà rimanere in memoria, e un riferimento al blocco di memoria occupato una volta che il nostro processo viene selezionato per entrare in esecuzione.

```
23  #classe che rappresenta il singolo processo
24  class Processo:
25      #COSTRUTTORE
26      def __init__(self, dimensione, ingresso, tEsecuzione):
27          self.dimensione = dimensione #rappresenta la dimensione del processo
28          self.ingresso = ingresso #rappresenta il tempo di clock in cui il processo deve (se puo') entrare in esecuzione
29          self.tEsecuzione = tEsecuzione #rappresenta quanti giri di clock necessita il processo per espletare le sue funzioni
30          self.bloccoOccupato = None #rappresenta il blocco di memoria a cui e' stato assegnato il processo
31
```

- **Log**: classe Python Log, all'interno della quale vengono gestiti i file di log e la loro stampa

```
49  #Classe per effettuare i print tutti assieme, senza dover scrivere i comandi piu' volte
50  class Log:
51      def __init__(self, filename, permission):
52          #Creo un file di log se non esiste. Altrimenti continuo a scrivere.
53          self.log = open(filename + ".txt", permission)
54
55
56      #Stampo il testo
57      def write(self, text):
58          #Print su shell
59          print text
60
61          #Log del testo su file senza colori
62          self.log.write(text + "\n")
63
```

3.3 Modalità d'uso

Le modalità di utilizzo di questa applicazione Python sono due, manuale ed automatica. Il menu che è stato implementato permette, una volta inserito il numero di simulazioni, di scegliere o singolarmente un determinato algoritmo tra First Fit, Best Fit e Worst Fit, o l'esecuzione automatica di tutti e tre gli algoritmi precedenti. In entrambi i casi, verranno visualizzati a video i risultati, i quali saranno riportati anche nel file di log.

4 Statistiche

L'applicazione, oltre ad eseguire i tre algoritmi, calcola anche le seguenti statistiche:

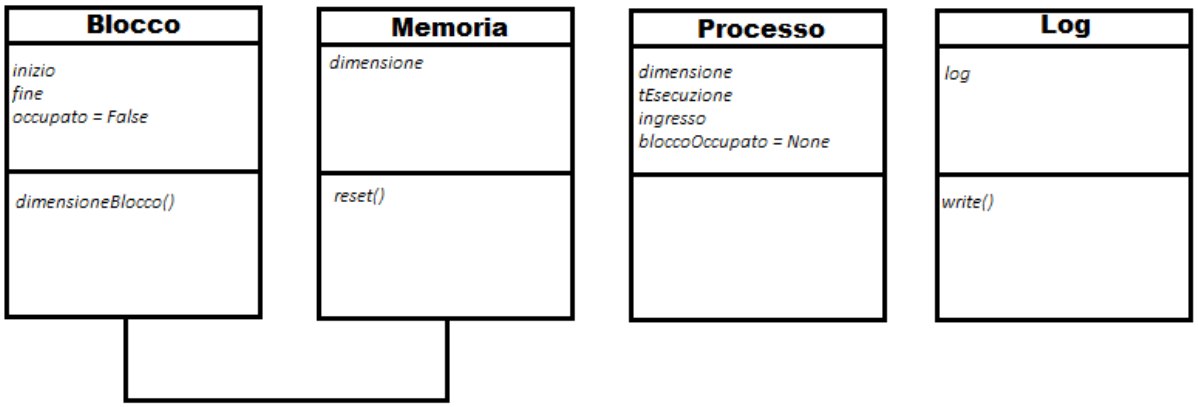
- **Confronti effettuati**: viene conteggiato il numero di confronti che vengono effettuati nella ricerca di un blocco che soddisfi i requisiti dell'algoritmo scelto, e che sia quindi adatto ad ospitare il processo "bersaglio". Questo servirà per verificare e stabilire quale dei tre algoritmi sia il più efficiente nelle simulazioni.
- **Frammentazione esterna**: si parla di frammentazione esterna quando lo spazio di memoria totale è sufficiente a soddisfare una richiesta ma non è contiguo (per ovviare a questa situazione sarebbe possibile fare una deframmentazione, cioè unire i vari blocchi non contigui in un unico blocco, ma questo comporta un notevole dispendio di risorse). La frammentazione esterna corrente viene accumulata con un contatore ad ogni iterazione di simulazione, e al terminare di quest'ultima viene divisa per il numero di iterazioni effettuate e poi mostrata a video all'utente. La frammentazione media viene calcolata ad ogni iterazione con la seguente formula

$$\text{FrammentazioneMedia} = 1.0 - \frac{\text{bloccoLiberoMax}}{\text{totMemoriaLibera}}$$

5 Diagrammi UML

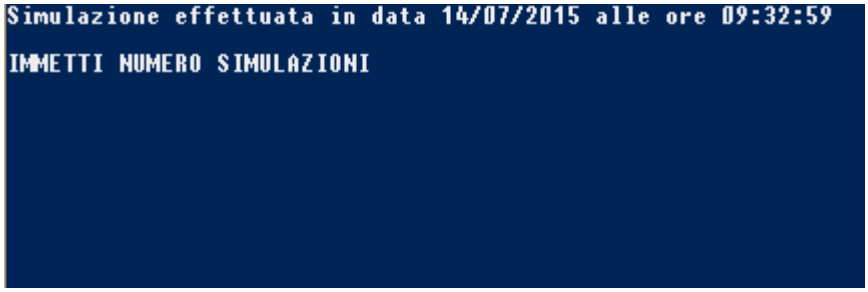
5.1 Class Diagram

Di seguito il class diagram UML delle classi contenute nel progetto.

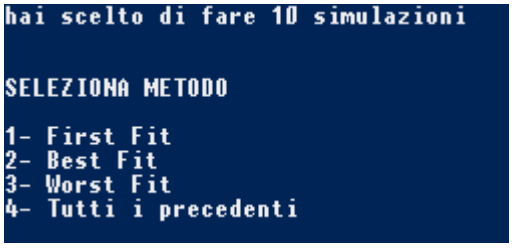


6 Manuale d'uso

All'avvio dell'applicazione, che si avvia con il comando "python2 scheduling.py", viene visualizzata la data e l'ora di inizio simulazione e viene chiesto all'utente di inserire il numero di simulazioni da effettuare



Successivamente viene chiesto all'utente di scegliere se utilizzare una singola tecnica o tutte e tre insieme



Il sistema, una volta inserito il valore, stampa prima tutti i processi in attesa e poi inizia a eseguire l'algoritmo.
La stampa della memoria avviene quando termina un processo, e si caratterizza per il simbolo pipe (|) nel caso in cui il blocco stampato è occupato, e per il simbolo cancelletto (#) se il blocco è libero, e viene stampata anche la sua dimensione.

```
processo terminato, stampa del nuovo stato di memoria
stampa della memoria!

¦ 1625 ¦
# 5518 #
¦ 3158 ¦
¦ 4694 ¦
# 369 #
¦ 752 ¦
¦ 3510 ¦
# 854 #
```

Ad ogni ciclo, ne viene stampato l'indice ed anche quanti processi dovranno prima o poi terminare (lifetime)

```
clock: 74

lifetime: 14
```

Se il processo pronto non riesce ad avere un blocco assegnato, viene calcolata la frammentazione

```
entro nella verifica della frammentazione
questo e' il blocco max: 2988dimensione blocchi: 4211
2988  4211
frammentazione esterna: 0.290429826645
```

7 Risultati

I risultati conseguiti si possono considerare allineati alle aspettative, a prescindere dal numero di simulazioni effettuati: difatti l'algoritmo First Fit risulta essere quello con il minor numero di confronti effettuati, e i Best e Worst Fit sono invece simili tra loro.
Il valore medio della frammentazione invece va a premiare l'algoritmo di Best Fit, seguito da First e Worst Fit. Il maggior numero di allocazioni fallite è invece appannaggio del Worst Fit.