

Documentazione

GestMem

(Simulatore Allocazione Dinamica Processi)

Specifiche Progetto

Si deve realizzare un applicativo PERL che simuli il funzionamento di un gestore della memoria. Il gestore deve accettare come argomenti passati da linea di comando i seguenti dati:

- numero massimo di processi
- dimensione della memoria centrale dedicata ai processi
- attivazione (o meno) della swap area
- dimensione massima della swap area
- dimensione massima di un processo
- dimensione minima di un processo
- algoritmo di allocazione dei processi
- tempo minimo di permanenza in memoria di un processo
- tempo massimo di permanenza in memoria di un processo

Una volta passati tutti questi parametri al gestore (alcuni dati possono essere considerati di default), il gestore deve generare un numero random da interpretare come numero corrente dei processi, per ognuno dei quali stabilisce il tempo di permanenza, l'istante di arrivo in memoria e la dimensione. Sulla scorta di questi dati, il gestore deve provvedere ad allocare i processi, eventualmente impiegare la swap area, deallocare i processi man mano che termina il periodo di permanenza in memoria.

I tempi citati in questo progetto (tempi di permanenza e istante di arrivo) sono da considerarsi di riferimento, nel senso che non si richiede l'impiego di unità di misura. Nel corso della sua elaborazione, il gestore deve mostrare a video (e su un apposito file di log) ogni operazione di allocazione e deallocazione. La scelta della struttura di gestione dello stato corrente della memoria (bitmap, lista concatenata, buddy system) è facoltativa.

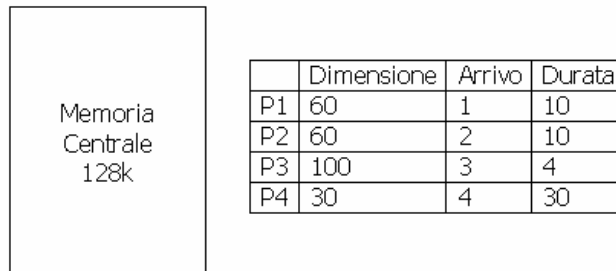
Nel corso dell'elaborazione deve però essere visionabile, e deve anche essere stampata su file.

Lo studente può realizzare l'applicativo corredandolo di interfaccia grafica mediante l'impiego delle librerie perl-tk o perl-gtk.

Premessa

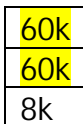
Il gestore della memoria in un sistema multitasking a partizionamento dinamico della memoria permette di risolvere il problema della frammentazione interna (partizionamento statico) e ridurre, attraverso appositi algoritmi di allocazione e compattazione della memoria, la frammentazione esterna.

Ipotizzando un sistema con una memoria centrale di 128 k e 4 processi di dimensione e



istante di arrivo casuali avremo la seguente situazione :

Dopo 2 istanti di tempo avremo :



Al terzo istante di tempo il P3 dovrà essere allocato nell'area di swap o, se non attiva, attendere fino alla conclusione di entrambi i processi.

Una volta che P1 e P2 avranno terminato i blocchi di memoria rimasti liberi (60,60,8) verranno fusi ripristinando la situazione iniziale. Questo è possibile quando i blocchi liberi sono contigui.

La swapping dei processi avviene quando un processo non trova spazio in memoria centrale. Quando si libera la memoria il processo verrà riallocato nel primo blocco disponibile.

Man mano che avviene l'allocazione dei processi possono crearsi delle zone di memoria libere non contigue (hole). Questa problematica viene risolta attraverso degli algoritmi di allocazione in memoria più o meno efficienti che scelgono l'area di memoria libera più adatta ad allocare il processo:

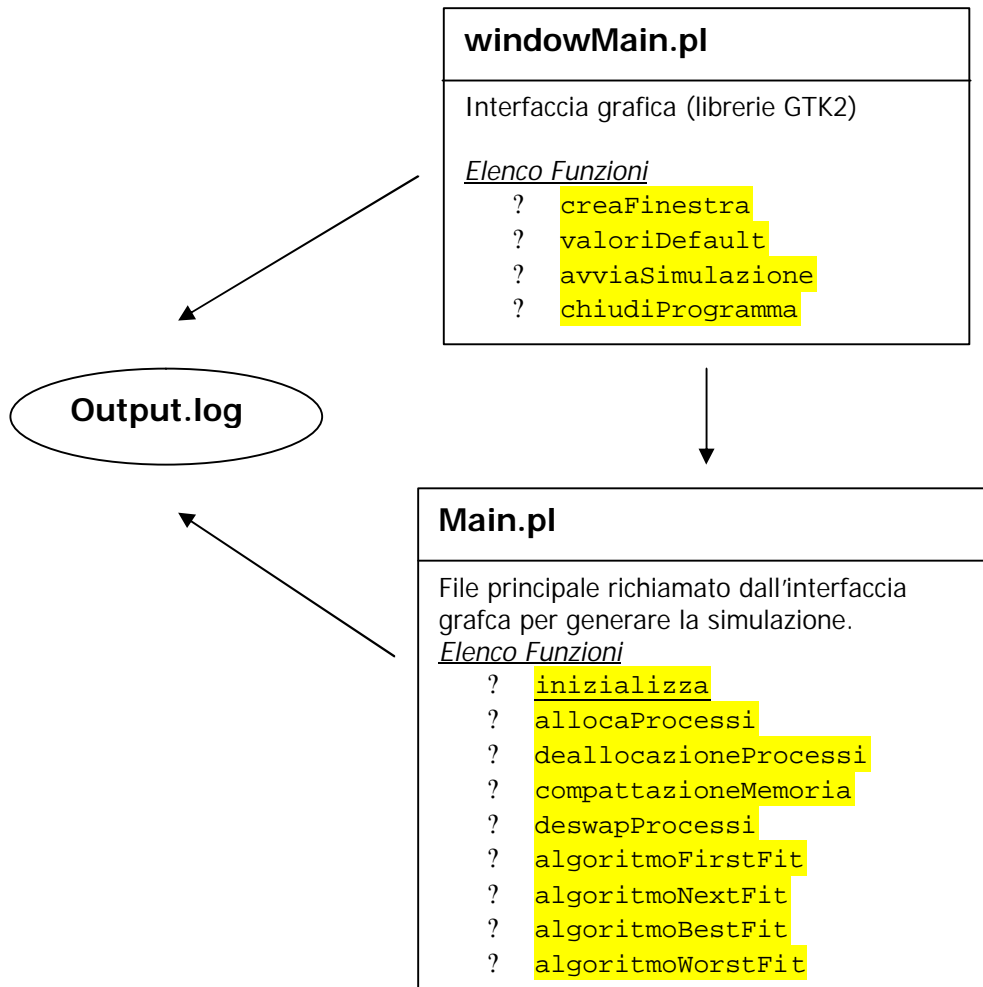
First fit: mantengo una lista delle zone libere in ordine di indirizzo; all'arrivo del processo lo si alloca nel primo spazio libero sufficiente.

Next fit: mantengo una lista delle zone libere in ordine di indirizzo; all'arrivo del processo lo si alloca nel primo spazio libero sufficiente. A differenza del First Fit la ricerca avviene dall'ultima posizione trovata.

Best fit: mantengo una lista delle zone libere in ordine crescente di dimensione; all'arrivo del processo lo si alloca nel primo spazio libero sufficiente (quindi nel più piccolo spazio sufficiente).

Worst fit: mantengo una lista delle zone libere in ordine decrescente di dimensione; all'arrivo del processo lo si alloca nel primo spazio libero (se sufficiente).

Struttura Progetto



Main.pl utilizza 4 packages per la creazione di oggetti.

GestoreProcessi.pm per immagazzinare gli argomenti forniti, per generare i processi e ordinarli, per estrarre i Blocchi di memoria liberi e per loggare gli eventi.

Processo.pm per creare n oggetti Processo da allocare in degli array (Liste)

MemoryBlock.pm per creare blocchi di memoria per creare e gestire la memoria centrale e di swap.

LogClass.pm per tenere traccia degli step della simulazione.

<p>GestoreProcessi.pm</p> <p>Classe per la memorizzazione degli argomenti e con funzioni per la gestione dei processi</p> <p><u>Proprietà Classe</u> nmaxProcessi dimMemoria dimSwaparea dimMaxproc dimMinproc algoritmo tempMinpermanenza tempMaxpermanenza</p> <p><u>Elenco Funzioni</u> ? genera_numProcessi ? insertion_sort ? estraiBloccoPiccolo ? estraiBloccoGrande ? logProprieta ? logMemoria ? logProcessi</p>	<p>Processo.pm</p> <p>Classe per la creazione di una Entità Processo con funzioni di generazione casuale dei campi.</p> <p><u>Proprietà Classe</u> identificativo swapping indirizzomemoria tempopermanenza istantediarrivo dimensione</p> <p><u>Elenco Funzioni</u> ? genera_Istantearrivo ? genera_Tempopermanenza ? genera_Dimensione</p>
<p>MemoryBlock.pm</p> <p>Classe per la creazione di una Entità Blocco di Memoria utilizzata per l'allocazione dei processi</p> <p><u>Proprietà Classe</u> indirizzo bitdistato dimensione</p> <p><u>Elenco Funzioni</u></p>	<p>LogClass.pm</p> <p>Classe per la stampa dei messaggi di log dell'applicazione.</p> <p><u>Proprietà Classe</u> stringLog</p> <p><u>Elenco Funzioni</u> ? pushLog ? printLog ? resetLog</p>

Elenco Funzioni

windowMain.pl

- ? **creaFinestra**
Disegna la finestra principale del programma con l'ausilio delle librerie GTK2
- ? **valoriDefault**
Valorizza le text di input con i valori di default .
- ? **avviaSimulazione**
Lancia il comando "perl Main.pl > ./output.log" .
Disegna la finestra di output .
Apre il file output.log e lo mostra a video .
- ? **chiudiProgramma**
Esce dal programma .

Main.pl

- ? **inizializza**
Acquisisce gli argomenti da riga di comando .
Genera la coda dei processi in base agli argomenti forniti .
Genera i blocchi iniziali di memoria centrale e swap .
- ? **allocaProcessi**
Start dell'allocazione in memoria centrale o swap(se attiva)
l'allocazione avviene in maniera dinamica in base all'algoritmo passato per argomento .
B : Best Fit - F : First Fit - N : Next Fit - W : Worst Fit

Il ciclo itera l'array dei processi verificando :

1 - Controlla se i processi allocati hanno terminato il loro tempo di permanenza
nel caso positivo de-alloca i processi e unisce eventuali aree di memoria contigue (Compattazione) .

2 - Controlla se ci sono processi nell'area di swap
nel caso positivo controlla se c'è spazio in memoria centrale sufficiente
nel caso positivo sposta il processo in memoria centrale, dealloca eventuali processi e compatta la memoria .

3 - Confronta l'istante di tempo con l'istante di arrivo del processo
a - Nel caso `istanteditempo==istantediarriivo`
a1 - Applica l'algoritmo di allocazione passato per argomento
Se l'allocazione non va a buon fine viene incrementato l'istante di arrivo del processo
a2 - Incrementa gli istanti di tempo

b - Nel Caso `istante di tempo>istante di arrivo`
(Processi con istante di arrivo uguali o proveniente da swap)
b1 - Incrementa l'istante di arrivo del processo
b2 - Applica l'algoritmo di allocazione passato per argomento
Se l'allocazione non va a buon fine viene incrementato l'istante di arrivo del processo
b3 - Incrementa gli istanti di tempo

? **deallocazioneProcessi**

L'array dei processi, oltre a simulare la coda dei processi, viene utilizzato per de-allocare i processi. Questo perché ogni processo possiede il campo "indirizzo in memoria".

Quando un processo viene allocato in memoria l'oggetto che lo identifica viene spostato nell'array degli indirizzi e, quando termina il suo tempo di permanenza, oltre a liberare la memoria (Bit stato=0) viene eliminato dalla lista degli indirizzi.

? **compattazioneMemoria**

La Memoria Centrale viene simulata con un array di Oggetti dove ogni oggetto contiene i campi :

Indirizzo - Bit di Stato - Dimensione

ogni oggetto simula una zona di memoria . quando due zone contigue si liberano vengono fuse in una unica zona più grande.

? **deswapProcessi**

Gestione SWAPPING

Se un processo in coda non trova aree libere finisce in swap.

Quando si liberano aree in memoria centrale il processo abbandona l'area di swap.

? **algoritmoFirstFit**

Cerca la prima area di memoria centrale libera sufficiente in ordine di Indirizzo.

? **algoritmoNextFit**

Cerca la prima area di memoria centrale libera sufficiente in ordine di Indirizzo iniziando la ricerca dall'ultima posizione in cui ha trovato Un blocco di memoria libero.

? **algoritmoBestFit**

Estraggo il blocco di memoria libero più piccolo avendo ordinato i blocchi di memoria liberi per dimensione crescente.

? **algoritmoWorstFit**

Estraggo il blocco di memoria libero sufficientemente grande avendo ordinato i blocchi di memoria liberi per dimensione decrescente.

GestoreProcessi.pm

? **genera_numProcessi**

Genera, in base all'argomento passato, la quantità di processi da generare.

? **insertion_sort**

Dopo aver generato l'array di oggetti Processo viene richiamata questa funzione che implementa l'algoritmo di ordinamento "insertion sort" per ordinare in base all'istante di arrivo i Processi.

? **estraiBloccoPiccolo**

Richiamata dall'algoritmo Best Fit, ordina le aree libere per dimensione crescente ed estrae il primo blocco sufficientemente grande.

? **estraiBloccoGrande**

Richiamata dall'algoritmo Worst Fit, ordina le aree libere per dimensione decrescente ed estrae il primo blocco sufficientemente grande.

- ? **logProprieta**
Stampa gli argomenti forniti e il numero di processi generato.
- ? **logMemoria**
Stampa la situazione della memoria centrale e swap in un determinato istante di tempo.
- ? **logProcessi**
Stampa l'elenco dei processi generati.

Processo.pm

- ? **genera_Istantearrivo**
Genera in maniera casuale, in un range compreso tra 1 a 100, l'istante di arrivo del processo.
- ? **genera_Tempopermanenza**
Genera in maniera casuale, in un range ottenuto tramite gli argomenti Forniti, il tempo di permanenza del processo.
- ? **genera_Dimensione**
Genera in maniera casuale, in un range ottenuto tramite gli argomenti Forniti, la dimensione del processo.

LogClass.pm

- ? **pushLog**
Memorizza in una variabile il messaggio di log fornito.
- ? **printLog**
Stampa la variabile valorizzata attraverso la funzione pushLog
- ? **resetLog**
Setta a vuoto la variabile contenitore dei messaggi.