GERARDO BOCHICCHIO

PROCESS SCHEDULER

SIMULATORE DI VARIE POLITICHE DI SCHEDULAZIONE DEI PROCESSI

SISTEMI OPERATIVI – SCIENZE E TECNOLOGIE INFORMATICHE – UNIVERSITÀ DEGLI STUDI DELLA BASILICATA

SCHEDULING DEI PROCESSI

Quando la CPU entra in uno stato di inattività, il sistema operativo sceglie quale dei processi presente nella ready queue debba essere eseguito.

Lo scheduling è un'operazione **molto importante per il corretto ed efficiente funzionamento** del calcolatore. Infatti non solo consente di eseguire più programmi concorrentemente, ma consente anche di migliorare l'utilizzo del processore.

L'obiettivo dello scheduling è quello di:

- **massimizzare il throughput**, ovvero la produttività dell'intero sistema, minimizzando i tempi in cui la risorsa è inutilizzata;
- cercare l'ordinamento di richieste che **minimizza il rapporto tra tempo di completamento e tempo di "turnaround";**
- **evitare fenomeni indesiderati** come la starvation ovvero "l'attesa eterna" di alcune richieste, verificabile in determinate condizioni;
- dare all'utente del sistema **la percezione che le richieste vengano soddisfatte contemporaneamente.**

SCHEDULING PREEMPTIVE E NON-PREEMPTIVE

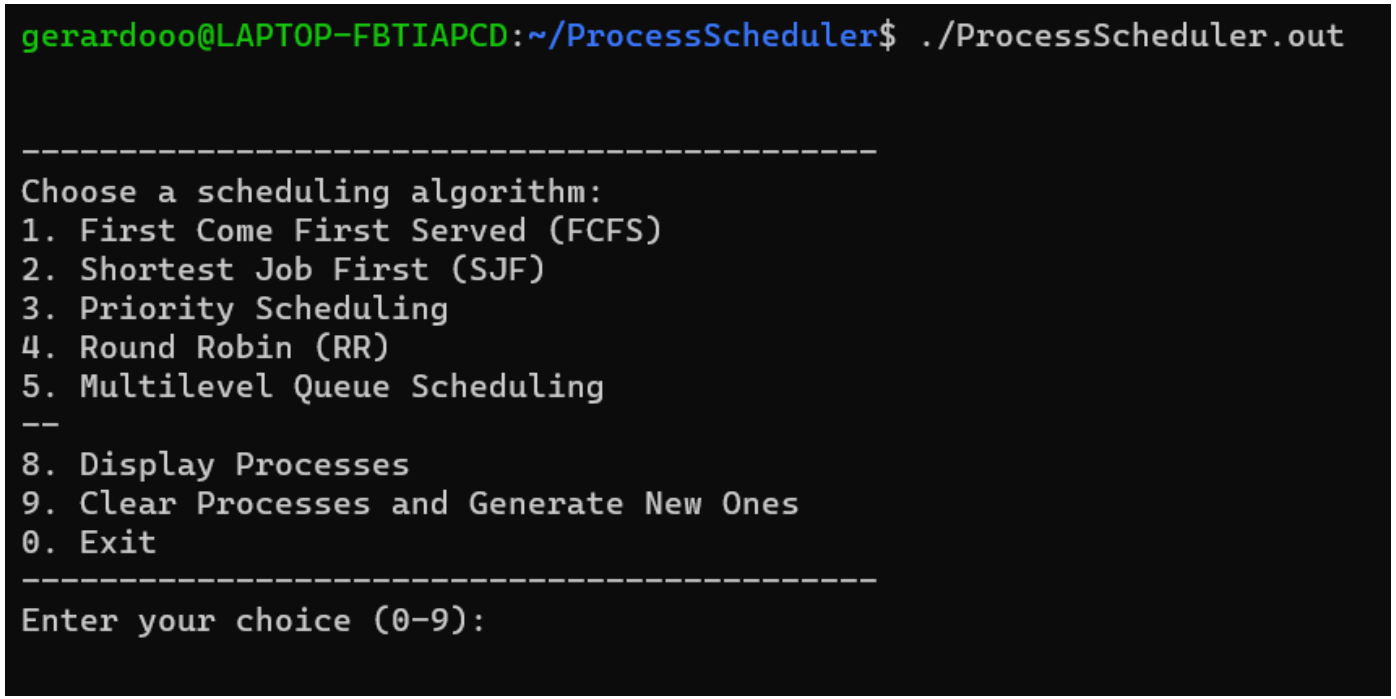
Lo scheduling dei processi può essere classificato in due categorie principali: **preemptive** (con diritto di prelazione) e **non-preemptive** (senza diritto di prelazione o cooperativo).

- Lo scheduling preemptive permette al sistema operativo di interrompere un processo in esecuzione per assegnare la CPU a un altro processo più prioritario. Questo tipo di scheduling è cruciale per garantire che i processi ad alta priorità ricevano tempestivamente le risorse necessarie, migliorando così la reattività del sistema.
- Al contrario, nello scheduling non-preemptive, una volta che un processo ha iniziato la sua esecuzione, lo porterà a termine senza interruzioni. In questo scenario, i processi cooperano cedendo volontariamente il controllo della CPU una volta completato il loro compito o raggiunto un punto in cui possono attendere.

IL PROGETTO

Lo scheduling dei processi è una componente fondamentale dei sistemi operativi. Esso determina l'ordine in cui i processi vengono eseguiti sulla CPU. Ci sono diversi algoritmi di scheduling, ognuno con i propri vantaggi e svantaggi.

Il programma, realizzato in C++, fornisce un'interfaccia per testare e confrontare diversi algoritmi di scheduling dei processi in un ambiente simulato.



IL PROGETTO: COMPONENTI PRINCIPALI

Classe Process: Rappresenta un processo con un ID, un tempo di burst e una priorità.

generateProcesses: genera processi con valori casuali

displayProcesses: mostra i dettagli dei processi.

total_burst_time: Calcola il tempo di burst totale di tutti i processi.

first_come_first_served, **shortest_job_first**, **priority_scheduling**, **round_robin**, **multilevel_queue_scheduling**: simulano i rispettivi algoritmi di scheduling.

main: Funzione principale che permette all'utente di scegliere un algoritmo di scheduling, visualizzare i processi o generare nuovi processi. La funzione include una convalida dell’input.

IL PROGETTO: CLASSE PROCESS

Il progetto contiene una classe Process per rappresentare i processi. Ogni processo è caratterizzato da tre proprietà:

- **pid**: è l'identificativo del processo. Ogni processo ha un ID unico che lo distingue dagli altri processi.
- **burst_time**: il tempo che il processo richiede per completare la sua esecuzione. È una misura del tempo di CPU necessario per eseguire il processo.
- **priority**: rappresenta la priorità del processo. Una priorità più bassa indica una priorità maggiore, il che significa che il processo dovrebbe essere eseguito prima degli altri con una priorità inferiore.

```
class Process {
public:
    int pid;
    int burst_time;
    int priority;

    Process(int pid, int burst_time, int priority)
        : pid(pid), burst_time(burst_time), priority(priority) {}
};
```

IL PROGETTO: GENERAZIONE DEI PROCESSI

I processi vengono creati con la funzione **generateProcesses** che riceve in input il numero di processi da creare e inserire nel vettore Process. Ad ogni processo viene assegnato un ID, un burst_time casuale compreso tra 1 e 20 e una priority casuale compresa tra 1 e 3.

```
std::vector<Process> generateProcesses(int num_processes) {
    srand(time(nullptr));
    std::vector<Process> processes;

    for (int i = 0; i < num_processes; i++) {
        processes.emplace_back(i, rand() % 20 + 1, rand() % 3 + 1);
    }

    return processes;
}
```

```
ID      Burst Time      Priority
-------------------------------
0       8               1
1       16              3
2       19              1
3       9               2
4       17              3
5       12              3
6       9               3
7       2               1
8       3               2
9       20              1
```

ALGORITMI DI SCHEDULING

Esistono vari algoritmi di scheduling che tengono conto di varie esigenze e che possono essere più indicati in alcuni contesti piuttosto che in altri. La scelta dell'algoritmo da usare dipende da cinque principali criteri:

1. **Utilizzo del processore**: la CPU (ovvero il processore) deve essere attiva il più possibile, ovvero devono essere ridotti al minimo i possibili tempi morti.
2. **Throughput**: il numero di processi completati in una determinata quantità di tempo.
3. **Tempo di completamento**: il tempo che intercorre tra la sottomissione di un processo ed il completamento della sua esecuzione.
4. **Tempo d'attesa**: il tempo in cui un processo pronto per l'esecuzione rimane in attesa della CPU (waiting time).
5. **Tempo di risposta**: il tempo che trascorre tra la sottomissione del processo e l'ottenimento della prima risposta.

Un algoritmo di scheduling si pone i seguenti obiettivi:

- **Fairness** (equità): processi dello stesso tipo devono avere trattamenti simili
- **Balance** (bilanciamento): tutte le parti del sistema devono essere sfruttate

ALGORITMI DI SCHEDULING

In questo progetto sono stati presi in considerazione i seguenti algoritmi di scheduling dei processi.

1. First Come First Served (FCFS)
2. Shortest Job First (SJF)
3. Priority Scheduling
4. Round Robin (RR)
5. Multilevel Queue Scheduling (MQS)

Esiste una variante del MQS, il Multilevel Feedback Queue Scheduling.

FIRST-COME, FIRST-SERVED (FCFS)

FCFS è il più semplice algoritmo di scheduling. I processi vengono eseguiti nell'ordine in cui arrivano nella coda di attesa, e ogni processo deve attendere che il processo precedente sia completato prima di poter essere eseguito.

Nel codice Il tempo di attesa per ogni processo viene calcolato sommando i tempi di esecuzione dei processi che lo precedono. L'attesa media viene calcolata sommando tutti i tempi di attesa e dividendo per il numero di processi.

Caratteristiche:

- Non-Preemptive: Una volta che un processo inizia, continua fino al completamento.
- FIFO (First In, First Out): L'ordine di esecuzione è determinato dall'ordine di arrivo dei processi.
- Semplice da implementare: Richiede solo una coda FIFO.

FIRST-COME, FIRST-SERVED (FCFS)

```
FCFS Scheduling:
Process Burst Time      Priority        Waiting Time
0       8               1               0
1       16              3               8
2       19              1               24
3       9               2               43
4       17              3               52
5       12              3               69
6       9               3               81
7       2               1               90
8       3               2               92
9       20              1               95
Average Waiting Time: 55.4
```

SHORTEST-JOB-FIRST (SJF)

SJF è un algoritmo di scheduling che seleziona per l'esecuzione il processo con il tempo di CPU stimato più breve tra tutti i processi disponibili.

Nel codice, i processi vengono ordinati in base al loro burst_time utilizzando la funzione std::sort. Come nel FCFS, il tempo di attesa viene calcolato in base ai processi che precedono il processo corrente nella lista ordinata.

Caratteristiche:

- Non-Preemptive o Preemptive: SJF può essere implementato in entrambe le modalità. Nella versione preemptive, se arriva un nuovo processo con una durata stimata minore del tempo rimanente del processo corrente, il processo corrente viene interrotto.
- Priorità basata sulla durata: I processi con durata minore hanno la priorità più alta.

SHORTEST-JOB-FIRST (SJF)

```
SJF Scheduling:
Process Burst Time      Priority        Waiting Time
7       2               1               0
8       3               2               2
0       8               1               5
3       9               2               13
6       9               3               22
5       12              3               31
1       16              3               43
4       17              3               59
2       19              1               76
9       20              1               95
Average Waiting Time: 34.6
```

PRIORITY SCHEDULING

Priority Scheduling è un metodo di scheduling dei processi basato su priorità, dove a ogni processo viene assegnata una priorità. I processi con priorità più alta vengono eseguiti prima di quelli con priorità più bassa.

Nel codice, i processi vengono ordinati in base alla loro priorità utilizzando una funzione lambda e std::sort. Il tempo di attesa viene calcolato in modo simile a FCFS e SJF, ma utilizzando l'ordine di priorità.

Caratteristiche:

- Preemptive o Non-Preemptive: Può essere implementato in entrambe le modalità. Nella versione preemptive, un processo in esecuzione può essere interrotto se arriva un processo con priorità più alta.
- Assegnazione delle Priorità: Le priorità possono essere assegnate staticamente o dinamicamente.

PRIORITY SCHEDULING

```
Priority Scheduling:
Process Burst Time      Priority        Waiting Time
0       8               1               0
2       19              1               8
7       2               1               27
9       20              1               29
3       9               2               49
8       3               2               58
1       16              3               61
4       17              3               77
5       12              3               94
6       9               3               106
Average Waiting Time: 50.9
```

ROUND ROBIN (RR)

Round Robin è un algoritmo di scheduling preemptive che assegna a ciascun processo un'unità di tempo fissata, chiamata QUANTUM (quanto di tempo), e cicla attraverso i processi in coda eseguendoli per un quanto di tempo alla volta.

Nel codice, il quantum è definito come QUANTUM. Il codice mantiene una lista di remaining_burst_time per ogni processo. Se il remaining_burst_time di un processo è maggiore del quantum, il processo viene eseguito per il quantum e poi rimesso in coda. Se è minore o uguale al quantum, il processo viene eseguito fino al completamento.

Caratteristiche:

- Preemptive: Un processo viene eseguito per un quanto di tempo e poi rimesse in coda se non è completato.
- Quanto di Tempo: L'unità di tempo assegnata a ciascun processo, la cui scelta è cruciale per l'efficienza dell'algoritmo.

ROUND ROBIN (RR)

```
Round Robin Scheduling:
Process Burst Time      Priority        Waiting Time
0       8               1               33
1       16              3               79
2       19              1               91
3       9               2               69
4       17              3               94
5       12              3               74
6       9               3               78
7       2               1               28
8       3               2               30
9       20              1               95
Average Waiting Time: 67.1
```

MULTILEVEL QUEUE SCHEDULING (MQS)

Il Multi-Level Queue Scheduling è un algoritmo che organizza i processi in diverse code di priorità, ognuna con il proprio algoritmo di scheduling, permettendo così una gestione differenziata basata sul tipo di processo o altri criteri.

Nel codice, ci sono tre code. La prima coda utilizza Round Robin, la seconda FCFS e la terza SJF. I processi vengono distribuiti casualmente tra le code. Ogni coda viene eseguita in sequenza, il che significa che tutti i processi nella prima coda devono essere completati prima di passare alla seconda coda, e così via.

Caratteristiche:

- Code Multiple: I processi sono suddivisi in diverse code basate su criteri specifici come la priorità, la tipologia di processo, ecc.
- Algoritmi Diversi: Ogni coda può avere un algoritmo di scheduling differente (es. FCFS, SJF, RR, ecc.).

MULTILEVEL QUEUE SCHEDULING (MQS)

```
-----------------------------------------
Coda 0: Round Robin.
Round Robin Scheduling:
Process Burst Time      Priority        Waiting Time
1       16              3               36
2       19              1               44
4       17              3               47
5       12              3               36
Average Waiting Time: 40.75
-----------------------------------------
Coda 1: First Come First Served (FCFS).
FCFS Scheduling:
Process Burst Time      Priority        Waiting Time
0       8               1               0
7       2               1               8
8       3               2               10
9       20              1               13
Average Waiting Time: 7.75
Average Waiting Time (including the previous queue): 263.75
-----------------------------------------
Coda 2: Shortest Job First (SJF).
SJF Scheduling:
Process Burst Time      Priority        Waiting Time
3       9               2               0
6       9               3               9
Average Waiting Time: 4.5
Average Waiting Time (including the previous queues): 326.5
-----------------------------------------
Average Waiting Time: 210.333
```

MULTILEVEL FEEDBACK QUEUE SCHEDULING (MFQS)

Esiste una variante del Multilevel Queue Scheduling (MQS), si tratta del Multilevel Feedback Queue Scheduling (MFQS). A differenza del MQS, questo algoritmo permette ai processi di spostarsi tra le code se il loro comportamento di esecuzione cambia, ad esempio, se necessitano di più o meno tempo CPU di quanto previsto.

Caratteristiche:

- Code Multiple: I processi sono suddivisi in diverse code basate sulla priorità, con la possibilità di spostarsi tra code differenti.
- Feedback: L'algoritmo adatta la priorità dei processi in base al loro comportamento e stime di burst di CPU.
- Preemptive: MLFQ è un algoritmo preemptive che può interrompere un processo in esecuzione se arriva un processo con priorità più alta.

L'algoritmo MFQS non è stato implementato poiché la rappresentazione delle code dinamiche e l'uso di un criterio casuale per lo spostamento dei processi tra le code avrebbero impedito una chiara dimostrazione del suo funzionamento.

PROCESS SCHEDULER

REPOSITORY GIT DEL PROGETTO

HTTPS://GITHUB.COM/GBOCHICCHIO/PROCESSSCHEDULER

GERARDO BOCHICCHIO

SISTEMI OPERATIVI – SCIENZE E TECNOLOGIE INFORMATICHE – UNIVERSITÀ DEGLI STUDI DELLA BASILICATA