**MFQ Scheduling Simulation**

결과 보고서

김세란

2018310520

**<목차>**

**1. 설계/구현 내용 설명**

**1) 설계 : RULE 규정**

**2) 구현 : 데이터, 알고리즘, 예외처리**

**2. 다양한 입력에 대한 실행결과**

**1) 다양한 입력**

**2) 실행 결과 : 스케줄링 과정 , 결과 출력 형식**

**1.설계/구현 내용 설명**

1. **설계**

MFQ scheduling은 process의 특성 (CPU/IO bound)에 맞게 우선순위를 조정하는 scheduling기법이므로 RULE을 다음과 같이 규정하였다.

RULE 규정:

1. 각 프로세스는 최초에 각자에게 지정된 queue로 진입하며, 최초 진입 queue에 의해

결정됨.

2.Qi에서 스케줄받아 실행하고 해당 Queue의 time quantum을 소모할 경우

i가 0,1,2 라면 Qi+1로 진입.

i가 3 이라면 Q3로 진입.

3.Qi에서 스케줄받아 실행하고 IO burst 마치고 wakeup 되는 경우 Qi-1로 진입

i가 1,2,3 라면 Qi+1로 진입.

i가 0 이라면 Q0로 진입

4.우선순위는 Q0>Q1>Q2>Q3 순이며, 스케줄링은 항상 높은 우선순위의 queue에서부터 이루어진다.

5. Q0,Q1,Q2는 각각 time quantum 2,4,8을 갖는 RR scheduling 기법을 사용하며, Q3은 time quantum이 무한한(없는) FCFS scheduling 기법을 사용한다.

1. **구현**

|  |  |
| --- | --- |
| **데이터** | |
| **Process 구조체** | **전역변수 TIME:** CPU TIME의미, process의 AT,TT,BT,WT등을 계산하기 위한 전역 변수  **Pid,cycle,BTlist**: 입력받은 process 정보를 저장  **AT**: 각 큐 기준으로 인큐 할 때 도착시간을 업데이트 해줌  **TT**: -초기AT+terminate **TIME**로 계산  **BT**:총 CPU BT, 나중에 WT계산하기 위함  **WT**:TT-BT로 계산  **BTlist**: CPU BT/IO BT  **Cycle**: cycle 수 에 따라 공간을 할당받음 |
| Process정보를 초기화하여 각 RQ에 인큐    **NofP**(process개수)만큼 프로세스 구조체 생성    RQ 별로  안에 들어있는 process 정보(pid,큐AT,현재BTlist) 프린트 |
| **Queue 구조체: process를 관리하며,**  **Front에는 각 큐 에 대한 process의 AT로 하여**  **AT가 가장 작은(우선순위가 가장 높은) process가 위치한다.** | **Pnum:** quueue안에 있는 process 개수  인큐 디큐 시 업데이트된다.  **QID:** RQ id, 0,1,2,3이 있다. |
| 큐 초기화, pnum=0;    Pnum++;  \*\*\*front에 AT가 낮은 프로세스가 오도록 정렬하여  Front에서 Dispatch 할 수 있도록 한다.    Pnum--;;  이후 프로세스 종료를 의미하거나 다른큐로 인큐될 것임    Front,즉 AT가 가장 작아서(우선순위가 높다) dispatch되어야하는 프로세스를 가르킨다. |

|  |  |
| --- | --- |
| **스케줄링 알고리즘** | |
| **(\*):**    **(\*\*):** | 아래부터  Process: P  RQn:(Qn)라고 칭한다  CPU TIME: T라 한다.  **전체 구조:**  **CPU 기준으로 process가 모두 종료할 때까지 하나씩 우선 순위가 높은 queue를 기준으로 dispatch하고 각 queue의 기법에 맞게 스케줄링 한다.**  While(1)  미래에 실행될 P들까지 포함하여 Q가 모두 비었으면 종료  Dispatch 할 queue 선정**-(\*)**  선정했으면 BURST - **(\*\*)**  **(\*):**  Q가 모두 비지 않았을 때  각Q의 front P 들의 AT가 모두 T보다 작지 않으면  ->Q의 front P 중 AT가 가장 작은 것을 선정:  이 경우,  P들이 IO burst로 인해 아직Q에 도착하지 않아 Q들이 모두 비어있는 상태로,  가장 빠른 P가 IO Burst를 마치고 돌아올 때 까지 CPU는 대기한다. 따라서 T도 증가  T보다 큰 것이 있다면  -> 그중 Q의 우선순위가 가장 높은 P를 선정  **(\*\*):**  각 Q에서 dispatch할 P를 선정했다면, **다음과 같은 3가지 상태**로 구분하고 CPU/IO Burst 및 scheduling.  **1.PREEMPTION O**  CPU Burst  낮은 queue로 스케줄링  **2.TERMINATE**  CPU Burst  Process 종료  **3.SLEEP O**  CPU Burst  IO Burst  높은 queue로 스케줄링  각 상황에 맞게  BTlist 소모, TIME 증가,  및 AT계산하여 enQueue |

|  |  |
| --- | --- |
| **잘못된 input에 대한 예외처리** | |
| 텍스트이(가) 표시된 사진  자동 생성된 설명 | 잘못된 input에 대해서는 파일 읽기를 멈추고 프로그램을 종료한다. 예외 처리 조건을 출력하도록 구현하였다.  예외처리조건으로는  - 설정한 개수 이상의 process 개수(NofP) input //NofP=10!!  - 인자의 개수가 부족한 경우  - pid가 중복되는 경우  - 0,1,2,3외의 Qid를 입력한 경우  - 필요한 것 이상의 입력을 한 경우  - cycle이 자연수가 아닌 경우  가있다. |

**2.다양한 입력에 대한 실행결과**

|  |  |
| --- | --- |
| **다양한 입력에 대한 설계** | |
| While문에 조건을 보면  Program 종료를 위한 마지막 else 조건을 제외하고 총 11개의 case가 있다.  모든 process들이 종료할 때 까지  하나씩 dispatch하는 이 while문에서, 전체 11개의 case 중 몇 개의 case를 만족하는지를 보면,  총 2^12 =4096개의 입력으로 구분할 수 있을 것이다.  이후 어떤 case들을 만족하는 지를 보고 스케줄링 의 다양성을 고려하여 다양한 입력에 대한 결과를 보이겠다.  따라서 이를 알기 위해 int cs[11]에 각 case의 flag 값을 저장하여, 모든 프로세스가 마치고 이를 출력하도록 하였다.  아래에서는  1)우선 스케줄링이 정상 작동하는 지 보여주는 과정 프린트와 결과 프린트를 보이고,  2)이후 Scheduling이 원활하게 돌아가므로 3개의 입력과, 그 입력이 어떤 종류의 입력인지를 보이겠다. | |
| **1)스케줄링 과정 및 결과 출력 구현** | |
| **과정 출력** | **결과 출력** |
| **Process를 하나씩 dispatch 할 때 마다**  **RQ 내에 있는 process를**  **각각의 우선순위(기준: AT)에 맞게 정렬하여**  **Process의 AT 및 남은 Burst time list 를 출력한다** | **<Gantt Chart>**    **CLOCK: CPU TIME (초기값: 0) 으로**  **각 process 시작할 때의 시각을 의미한다.**  **이후 화살표는 TIME=1을 ‘--’에 대응하여**  **얼마나 CPU Burst를 오래 했는지 나타낸다.**  **<Process 별 PID, TT, WT 그리고 TT평균과 WT평균>** |
| **2) 다양한 입력과 그에 대한 결과** | |
| **입력1**  **Input.txt**    **다양성:** | |
| **실행결과:** | |
| **입력2:**  **Input.txt**    **다양성:** | |
| **실행결과:** | |
| **입력3:**  **Input.txt**    **다양성:** | |
| **실행결과:** | |

감사합니다.