R E P O R T

OS Term Project #1

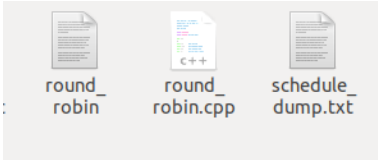
학과	컴퓨터교육과
교수님	류은석
학번	2020312163
이름	김지윤
제출일	2022.11.20

Term Project #1

1. Round-Robin Scheduling 개요
   - Round-Robin(RR) Scheduling 알고리즘은 시간분할 시스템을 위해 설계되었다. 선입선출 순서로 프로세스들이 디스패치되지만, 각 프로세스는 한정된 시간만큼만 실행된다. 이 시간 간격을 Quantum이라고 부른다. FIFO scheduling과 비슷하지만, 각 프로세스 전환에 선점이 더해졌다.

2. 기본 요구사항
   a. R-R Scheduling 방식에 따라 부모 프로세스로부터 10개의 자식 프로세스를 만들었다.
   b. Time Quantum을 10으로 설정했다.
   c. Run Queue와 Wait Queue를 생성하여, Run Queue에는 준비 상태의 Child process를, Wait Queue에 나머지를 넣는다.
   d. 자식 프로세스의 경우, 무한 루프를 실행한다.
   e. CPU Burst 값을 랜덤으로 생성했다.

3. 실행 과정 화면(리눅스, ubuntu 사용)




```
jiyun@jiyun-VirtualBox:~$ vi round_robin.cpp
jiyun@jiyun-VirtualBox:~$ g++ -o round_robin round_robin.cpp
jiyun@jiyun-VirtualBox:~$ ./round_robin
```

4. 구현 코드 - round_robin.cpp

```
#include <stdio.h>
#include <iostream>
#include <stdlib.h> // for rand and exit
#include <unistd.h> // for fork
#include <time.h>
#include <sys/wait.h>
#include <string>
#include <queue>
#include <algorithm> // for min/max

// for msg queue
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define _CRT_SECURE_NO_WARNINGS
#define PROCESS_NUM 10
#define QUANTUM 10

using namespace std;

// process(child) structer
typedef struct process {
  pid_t pid;
  float cpu_burst;
  float io_burst;
} PROCESS;

// PCB structer
typedef struct pcb {
  bool flag;
  float burst;
} PCB;

// MSG structer
typedef struct msg {
  long msgtype;
  PCB pcb;
} MSG;
```

```
int main() {
  // pid를 담을 배열
  PROCESS child[PROCESS_NUM];
  int status;
  int cnt = 0;

  printf("Parent Process ID: %d\n", getpid());

  for (int i = 0; i < PROCESS_NUM; i++) {

    pid_t forkProcess = fork(); // 자식 프로세스 생성

    child[i].pid = forkProcess;
    child[i].cpu_burst = rand() % 20 + 1; // CPU burst : 1 ~ 20

    // 자식 프로세스일 경우 무한 루프 실행
    if (forkProcess == 0) {
      int key_id; // message id
      long msgtype = getpid(); // Message type은 자식 프로세스 ID로 설정

      float cpu_burst = child[i].cpu_burst;

      key_t key = ftok("keyfile", 1234);
      MSG msg;
      msg.msgtype = msgtype;

      // Message 생성 및 에러 핸들링
      key_id = msgget(key, IPC_CREAT | 0666);


      if (key_id == -1) {
        perror("msgget() error!");
        exit(1);
      }

      // 자식 프로세스 시작
      do {
        // 부모 프로세스에서 Message 올 때까지 대기
        if (msgrcv(key_id, &msg, sizeof(PCB), msgtype, 0) != -1) {

          // 남아있는 CPU burst time이 Quantum보다 큰 경우
          if (cpu_burst > QUANTUM) {
            cpu_burst -= QUANTUM;
            sleep(QUANTUM);

            msg.pcb.flag = true;
            msg.pcb.burst = QUANTUM;

            msgsnd(key_id, &msg, sizeof(PCB), IPC_NOWAIT); // 부모 프로세스에게 Message 전달

          }

          // 남아 있는 CPU burst time이 Quantum보다 작은 경우
          else {
            cpu_burst = 0;
            sleep(cpu_burst);

            msg.pcb.flag = false;
            msg.pcb.burst = cpu_burst;

            msgsnd(key_id, &msg, sizeof(PCB), IPC_NOWAIT);
          }
        }
      } while (cpu_burst > 0);

      exit(0); // 자식 프로세스 정상 종료
    }
  }

  // 로그 파일 작성
  FILE* fp = fopen("schedule_dump.txt", "w"); // Write 모드

  // 작업 Queue 생성
  queue<long> waitQueue; // Wait QUEUE
  queue<long> runQueue; // Rune QUEUE
  float burst_time[PROCESS_NUM];
  float completion_time[PROCESS_NUM];
  float total_time = 0;
  //char s1[50];

  printf("\n[List of Run Status Processes]\n\n");
  fprintf(fp, "\n[List of Run Status Processes]\n\n");
  for (int i = 0; i < PROCESS_NUM; i++) {
    runQueue.push(i); // ENQUEUE

    printf("PID : \t%d\n", child[i].pid);
    printf("Arrival Time : \t%d\n", i);
    printf("Burst Time : \t%.2lf\n\n", child[i].cpu_burst);
    fprintf(fp, "PID : \t%d\n", child[i].pid);
```

```
  fprintf(fp, "Arrival Time : \t%d\n", i);
  fprintf(fp, "Burst Time : \t%.2lf\n\n", child[i].cpu_burst);

  burst_time[i] = child[i].cpu_burst;
}

key_t key = ftok("keyfile", 1234);
int key_id;
MSG msg;

// Message 생성
key_id = msgget(key, IPC_CREAT|0666);
if (key_id == -1) {
  perror("msgget() error!\n");
  exit(1);
}

// 부모 프로세스 시작
printf("\n[Process Execution Flow]\n");
fprintf(fp, "\n[Process Execution Flow]\n");

do {
  cnt++;
  printf("\n#%d -> ", cnt); // 몇 번째 Context Switching인지 출력
  fprintf(fp, "\n#%d -> ", cnt);

  long run = runQueue.front(); // 큐의 다음 순서 값 추출
  runQueue.pop(); // DEQUEUE

  printf("Running Process : P%ld - PID[%d] - Burst time : %.2lf\n", run + 1, child[run].pid, child[run].cpu_burst);
  fprintf(fp, "Running Process : P%ld - PID[%d] - Burst time : %.2lf\n", run + 1, child[run].pid, child[run].cpu_burst);

  msg.msgtype = child[run].pid; // msgtype 해당 자식 프로세스 pid 값으로 설정

  // 자식 프로세스에게 message 전송
  msgsnd(key_id, &msg, sizeof(PCB), IPC_NOWAIT);

  if (msgrcv(key_id, &msg, sizeof(PCB), child[run].pid, 0) != -1) {

    // burst가 남은 경우 다시 큐에 Enqueue
    if (msg.pcb.flag == true) {
      if (child[run].cpu_burst > 0) {
        runQueue.push(run);
        child[run].cpu_burst -= QUANTUM;
        total_time += QUANTUM;
      }
    }
    // burst가 남지 않았을 경우
    else {
      total_time += child[run].cpu_burst;
      child[run].cpu_burst = 0;
      completion_time[run] = total_time;
    }

    if (child[run].cpu_burst <= 0) {
      total_time += (child[run].cpu_burst);
      child[run].cpu_burst = 0;
      completion_time[run] = total_time;
    }
  }
} while (!runQueue.empty());

msgsnd(key_id, &msg, sizeof(PCB), IPC_NOWAIT); // 마지막 Message를 자식 프로세스에 전송
wait(&status); // 모든 자식 프로세스가 끝날 때까지 대기

printf("\nRun Queue EMPTY!\n");
fprintf(fp, "\nRun Queue EMPTY!\n");

float sumCompletion = 0.0;
float sumWaiting = 0.0;

float minCompletion = 123456789.0;
float maxCompletion = 0.0;
float minWaiting = 123456789.0;
float maxWaiting = 0.0;

printf("\n\n[Round Robin Scheduler]\n\n");
printf("PID\t\tBurst Time\tCompletion Time\t\tWaiting Time\n\n");
fprintf(fp, "\n\n[Round Robin Scheduler]\n\n");
fprintf(fp, "PID\t\tBurst Time\tCompletion Time\t\tWaiting Time\n\n");

// Completion Time과 Waiting Time의 최소, 최대, 합계를 계산
for (int i = 0; i < PROCESS_NUM; i++) {
  printf("%d\t\t%.2lf\t\t%.2lf\t\t\t%.2lf\n\n", child[i].pid, burst_time[i], completion_time[i], completion_time[i] - burst_time[i]);
  fprintf(fp, "%d\t\t%.2lf\t\t%.2lf\t\t\t%.2lf\n\n", child[i].pid, burst_time[i], completion_time[i], completion_time[i] - burst_time[i]);

  sumCompletion += completion_time[i];
  sumWaiting += (completion_time[i] - burst_time[i]);

  minCompletion = min(minCompletion, completion_time[i]);
  maxCompletion = max(maxCompletion, completion_time[i]);
```

```
    minWaiting = min(minWaiting, completion_time[i] - burst_time[i]);
    maxWaiting = max(maxWaiting, completion_time[i] - burst_time[i]);
  }

  // 위에서 계산한 값 출력
  printf("Min Completion Time : %.2f\n", minCompletion);
  printf("MAX Completion Time : %.2f\n", maxCompletion);
  printf("Avg Completion Time : %.2f\n\n", sumCompletion / PROCESS_NUM);
  fprintf(fp, "Min Completion Time : %.2f\n", minCompletion);
  fprintf(fp, "MAX Completion Time : %.2f\n", maxCompletion);
  fprintf(fp, "Avg Completion Time : %.2f\n\n", sumCompletion / PROCESS_NUM);

  printf("Min Waiting Time : %.2f\n", minWaiting);
  printf("MAX Waiting Time : %.2f\n", maxWaiting);
  printf("Avg Waiting Time : %.2f\n", sumWaiting / PROCESS_NUM);
  fprintf(fp, "Min Waiting Time : %.2f\n", minWaiting);
  fprintf(fp, "MAX Waiting Time : %.2f\n", maxWaiting);
  fprintf(fp, "Avg Waiting Time : %.2f\n", sumWaiting / PROCESS_NUM);

  fclose(fp);
  return 0;
}
```

제가 잘 모르는 분야이기도 하고, C++은 한 번도 배우지도 않았고, 써보지도 않았을 뿐더러, 프로젝트가 조금 어렵고 생소한 부분이 많았어서 구글링을 통해 여러 자료를 참고하고 공부해가며 작성하였습니다.

5. 실행 결과 파일 - schedule_dump.txt

```
[List of Run Status Processes]

PID :   4685
Arrival Time :  0
Burst Time :    4.00

PID :   4686
Arrival Time :  1
Burst Time :    7.00

PID :   4687
Arrival Time :  2
Burst Time :    18.00

PID :   4688
Arrival Time :  3
Burst Time :    16.00

PID :   4689
Arrival Time :  4
Burst Time :    14.00

PID :   4690
Arrival Time :  5
Burst Time :    16.00
```

```
PID :   4691
Arrival Time :  6
Burst Time :    7.00

PID :   4692
Arrival Time :  7
Burst Time :    13.00

PID :   4693
Arrival Time :  8
Burst Time :    10.00

PID :   4694
Arrival Time :  9
Burst Time :    2.00


[Process Execution Flow]

#1 -> Running Process : P1 - PID[4685] - Burst time : 4.00

#2 -> Running Process : P2 - PID[4686] - Burst time : 7.00

#3 -> Running Process : P3 - PID[4687] - Burst time : 18.00


#4 -> Running Process : P4 - PID[4688] - Burst time : 16.00

#5 -> Running Process : P5 - PID[4689] - Burst time : 14.00

#6 -> Running Process : P6 - PID[4690] - Burst time : 16.00

#7 -> Running Process : P7 - PID[4691] - Burst time : 7.00

#8 -> Running Process : P8 - PID[4692] - Burst time : 13.00

#9 -> Running Process : P9 - PID[4693] - Burst time : 10.00

#10 -> Running Process : P10 - PID[4694] - Burst time : 2.00

#11 -> Running Process : P3 - PID[4687] - Burst time : 8.00

#12 -> Running Process : P4 - PID[4688] - Burst time : 6.00

#13 -> Running Process : P5 - PID[4689] - Burst time : 4.00

#14 -> Running Process : P6 - PID[4690] - Burst time : 6.00

#15 -> Running Process : P8 - PID[4692] - Burst time : 3.00

Run Queue EMPTY!

[Round Robin Scheduler]
PID             Burst Time      Completion Time         Waiting Time

4685            4.00            4.00                    0.00

4686            7.00            11.00                   4.00

4687            18.00           88.00                   70.00

4688            16.00           94.00                   78.00

4689            14.00           98.00                   84.00

4690            16.00           104.00                  88.00

4691            7.00            58.00                   51.00

4692            13.00           107.00                  94.00

4693            10.00           78.00                   68.00

4694            2.00            80.00                   78.00
```

```
Min Completion Time : 4.00
MAX Completion Time : 107.00
Avg Completion Time : 72.20

Min Waiting Time : 0.00
MAX Waiting Time : 94.00
Avg Waiting Time : 61.50
```