과제1

202011376_조현서_운영체제(3199)_과제1_04.05

1. **과제의 목적**

- 자식 프로세스의 실행 순서를 Multi-Level Feedback Queue를 이용하여 scheduling해 봄으로 써 운영체제가 어떠한 방식으로 프로세스들을 관리하는지 이해할 수 있습니다.
- 스캐쥴러를 구현하면서 발생하는 정책 방식 등 여러가지 고려 사항들에 대해 고민해 볼 수 있습니다.

2. **개요**

- 입력으로 process와 timesilce의 수가 주어질 때, 한 개의 cpu가 있는 상황에서(한 번에 하나의 프로세스만 진행이 가능) cpu bound한 프로세스를 MLFQ를 이용하여 순서를 조정해주는 프로그램입니다.
- 기본적인 timeslice는 1초이며 time allotment가 2초가 되면 다음 priority queue로 넘어가며 10초마다 priority boost가 일어납니다.
- 단, priority boost가 발생하는 시점에 가장 큐에 있던 running 프로세스의 time allotment가 2에 도달했다면 그 process의 priority는 낮춥니다.
- 우선순위를 나타내는 큐들은 3개의 level로 구성되어 있습니다.

3. **디자인(Design)**

- 우선순위를 나타내는 큐들은 linked list로 구현하며, 각각의 node들은 PCB와 다음 node를 가리키는 포인터로 구성되어 있습니다.
- main함수에서 인자로 넘어온 수 만큼 fork를 하여 자식 프로세스를 생성한 후 인자로 'A'+i를 넘겨줍니다.(단, i는 자식의 순서)
- 부모 프로세스에서는 자식 프로세스의 node를 만들어(malloc) 최상위 큐의 맨 뒤에 삽입합니다.
- 5초 동안 자식 프로세스들이 모두 ku_app를 실행할 시간을 준 후에 1초마다 schedule handler를 호출해 줍니다.
- schedule handler에서는 signal을 이용하여 process들을 scheduling하며 timeslice가 입력으로 들어온 수만큼 돌면 모든 자식 process들을 reaping하고 메모리들을 free해 준 후 프로그램을 종료합니다.

4. **구현(Implementation)** - 주요 코드

- 구조체와 전역변수

```
typedef struct _PCB{
        int time_allotment;
        int pid;
}PCB;

typedef struct RQ_Node{
    PCB* data;
    struct RQ_Node* next;
}node;

int S = 0;
int PRIO = 2;   // Highest Priority
int TS;
node* READY_QUEUE[3]={NULL,NULL,NULL};//ready queues
node* RUNNING = NULL;
```

PCB에는 각각의 프로세스가 사용한 time_allotment와 signal을 보낼 때 사용하는 pid가 정의되어 있습니다.

linked list를 구성하는 node들은 PCB와 다음 node의 주소로 이루어져 있습니다.

linked list들은 모두 READY_QUEUE에 시작점들이 연결되어 있어 배열 형태로 접근이 가능합니다.

RUNNING는 현재 실행 중인 프로세스의 node를 가리킵니다.

- 메인 함수에서의 자식 프로세스 생성

```
// fork processes
      for(int i=0; i<n;i++){

              int child_pid;
              if ((child_pid=fork())==0){
                      char arg[2] = {'A','\0'};
                      arg[0] += i;
                      execl("./ku_app","ku_app",arg,(char*)0);
              }
              else{
                      // make PCB
                      PCB* block = malloc(sizeof(PCB));
                      block->time_allotment = 0;
                      block->pid = child_pid;

                      //make node
                      node* n_node =malloc(sizeof(node));
                      n_node->data = block;
                      n_node->next = NULL;

                      // add at ready_Queue[2](highest priority queue)
                      Insert_Node(&READY_QUEUE[2], n_node);
              }
      }
```

자식 프로세스에서 execl을 통해 ku_app을 실행시키고
부모 프로세스에서는 프로세스의 수가 입력 값으로 주어지기 때문에 PCB와 노드를 동적할당 해 줍니다. 이후 최상위 우선순위 큐에 넣어줍니다.

- schedule handler에서 priority boost 발생 시

```
if (S == 10){
        S = 0;
        if(PRIO == 2 && RUNNING->data->time_allotment == 2){
                RUNNING->data->time_allotment = 0;
                Put_Highest();
                Insert_Node(&READY_QUEUE[1],Delete_Node(&READY_QUEUE[2]));
        }
        else{
                // put running node at the end of the list
                Insert_Node(&READY_QUEUE[PRIO],Delete_Node(&READY_QUEUE[PRIO]));
                Put_Highest();
                PRIO = 2;
        }
}
```

Priority Boost가 발생한 경우, running process의 time allotment가 2가 되며 최상위 큐에 위치할 때에는 그 process의 time allotment를 0으로 만든 후 다른 프로세스들은 최상위 큐에 올리며 그 process만 두 번째 큐에 넣어줍니다.

이외의 경우에는 running process를 해당 큐의 맨 뒤에 넣은 후 모든 작업들을 맨 위의 작업 큐로 올려줍니다.

- schedule handler에서 종료 시

```
if (TS == 0)
        {
                // free allocated memory and kill children processes
                for (int i = 0; i<3;i++){
                        while(1){
                                node* temp = Delete_Node(&READY_QUEUE[i]);
                                if (temp == NULL)
                                        break;
                                else{
                                        // kill and reap
                                        kill(temp->data->pid,SIGKILL);
                                        waitpid(temp->data->pid,NULL,0);

                                        //free allocated memories
                                        free(temp->data);
                                        free(temp);
                                }
                        }
                }
                exit(0);
}
```

모든 큐에 있는(0,1,2) 노드들을 삭제한 후 그 노드의 프로세스를 종료시킨 후 reaping해 줍니다. 동시에 메모리도 회수해 줍니다.

- context switch

```
// context switching
kill(RUNNING->data->pid,SIGSTOP);
kill(READY_QUEUE[PRIO]->data->pid,SIGCONT);
RUNNING = READY_QUEUE[PRIO];
RUNNING->data->time_allotment += 1;
```

kill과 SIGSTOP. SOGCONT를 이용하여 context switch를 진행합니다. 실행중인 프로세스의 정보를 전달하며 time allotment도 바꿔줍니다.

5. **실행 예**
   - ku_mlfq 3 30 >& output1.txt (cat output1.txt - 그림으로 대체)

Q3 A⇒B⇒C⇒A⇒B⇒C ⇒B⇒C⇒A⇒B⇒C⇒A ⇒C⇒A⇒B⇒C⇒A⇒B

Q2 ⇒A⇒B⇒C⇒A ⇒B⇒C⇒A⇒B ⇒C⇒A⇒B⇒C

Q1

10초마다 priority boost가 일어나며 그때마다 running한 일들은 연결 리스트의 맨 뒤로 이동한 후 boost가 일어나 abc가 순서대로 출력이 됩니다.

- ku_mlfq 5 40 >& output2.txt (cat output2.txt - 그림으로 대체)

Q3 A⇒B⇒C⇒D⇒E⇒A⇒B⇒C⇒D⇒E ⇒A⇒B⇒C⇒D⇒A⇒B⇒C⇒D

Q2 (E만 Q2로 이동, 나머진 Q3로 이동) ⇒E⇒A

Q3 ⇒B⇒C⇒D⇒E⇒A⇒B⇒C⇒D⇒E⇒A ⇒B⇒C⇒D⇒E⇒B⇒C⇒D⇒E

Q2 (A만 Q2로 이동, 나머진 Q3로 이동) ⇒A⇒B

10초 후 E가 최상위 큐에서 time allotment가 2가 되며 실행이 끝날 때, priority boost가 일어나서 E는 아래로 이동하며 2번째 큐에 존재하던 ABCD는 최상위 큐에 올라와 순서대로 수행됩니다.

마찬가지로 30초 후에도 역시 BCDE는 위로 올라오기 때문에 BCDE가 순서대로 수행이 됩니다.

(개요 - 3 구현 확인)

함수

함수 이름	인자	설명
Insert_Node	-	target 우선 순위 큐(linked list)의 맨 뒤에 새로운 노드를 추가하는 함수 입니다.
	node** head	target 우선 순위 큐(linked list)의 주소를 전달해 줍니다. linked list의 처음 노드의 주소를 담은 배열의 주소를 넘겨줘야 하므로 투 포인터가 사용됩니다.
	node* n_node	추가할 노드를 나타냅니다.
	Return Value	void
Delete_Node	-	target 우선 순위 큐(linked list)의 맨 앞에 있는 노드를 삭제하는 함수입니다.
	node** head	target 우선 순위 큐(linked list)의 주소를 전달해 줍니다. linked list의 주소를 담은 배열의 주소를 넘겨줘야 하므로 투 포인터가 사용됩니다.
	Return Value	node*. 삭제를 성공한 경우에는 삭제한 노드의 주소를, 실패한 경우 NULL을 리턴합니다.
Put_Highest	-	Priority Boost에서 사용하는 함수로 2,3번째 우선 순위 큐에 있는 작업들의 time_allotment를 초기화 한 후 최상위 큐로 이동시킵니다.
	Return Value	void
schedule_handler	-	SIGALRM 호출 시 호출되는 signal handler로 전반적인 프로세스들의 스캐줄을 조정합니다.
main	-	
	int argc	입력으로 들어오는 인자의 수입니다.
	char* argv[]	입력으로 들어오는 인자로, 문자열 배열입니다.
	Return Value	성공하면 0, 이외에는 1과 함께 종료합니다.

(인자가 ‘-’ 인 경우 함수 자체의 설명을 의미합니다.)

붙임. ku_mlfq.c 끝.