<Assignment #1 : KU_CFS 보고서>

컴퓨터공학과 201811250 마지영

먼저 CFS의 Ready Queue를 구현하기 위해 Linked List를 사용하려고 하였습니다. 그래서 Linked List를 구현하기 위해 필요한 Node structure를 먼저 구현하였습니다. 이 Node는 프로세스의 pid값을 갖는 value 변수, nice 값을 갖는 nice 변수, v-runtime을 기록할 vrun 변수, 다음 Node를 나타내는 next 변수, 그리고 출력할 알파벳의 값을 갖는 c 변수를 가지고 있습니다.

그리고 Linked List structure를 선언해서 이를 이용하는 방법을 생각해보았습니다. 이 방법에 따르면 먼저 Linked List structure를 선언하고, main에서 Linked List 구조체 변수를 선언하거나 전역변수로 Linked List 구조체 변수를 선언하여 사용해야합니다. 전자처럼 할 경우, Linked List와 관련된 모든 함수의 parameter에 Linked List 변수를 넣어야 하기 때문에, 상당히 불편할 것이라고 생각하였습니다. 후자의 경우에는 모든 함수의 parameter에 Linekd List 변수를 넣어야 하는 수고스러움은 없지만, Node에 접근하려면 한 단계 더 들어가야 하기 때문에 그냥 Linked List가 있다고 생각하고, 이 Linked List의 head Node를 전역 변수로 선언하는 방법 또한 생각해보았습니다. 하지만, 추후에 수정할 부분이 생기는 경우를 대비해, Linked List 구조체를 간단하게, head node만 가질 수 있게 구현을 하였고, 이를 ready queue로 사용하기 위해 전역변수로 선언을 해주었습니다.

그리고 Node의 구조체 포인터 변수인 now와 before을 전역변수로 선언하였습니다. 이는, SIGALRM의 signal handler인 catcher 함수가 계속해서 방금까지 실행한 프로세스와 앞으로 실행할 프로세스를 계속해서 업데이트해가면서 사용하기 위해서 선언하였습니다. 그리고 정수형 변수인 timecount도 전역변수로 선언하였는데, 이 또한 SIGALRM이 발생한 횟수를 signal handler인 catcher에서 업데이트하고, 이를 main함수에서도 사용하기 위해 이 방법을 택하였습니다.

main에서는 제일 먼저 전역변수로 선언한 ready queue, 즉, readyQ 변수를 malloc을 이용해 메모리를 할당해주었습니다. 메모리 크기는 sizeof 함수를 이용하여 Linked List 구조체만큼 할당해주었습니다. 그리고 프로그램 실행 시, 입력받은 값들을 정리하였습니다. argv[6]에 위치한 time slice의 개수는 atoi함수를 이용하여 정수로 변환하여 time_slices 변수에 넣어주었고, nice 크기에 따른 프로세스 개수들은 크기가 5인 정수형 배열인 n에 for문과 atoi 함수를 이용하여 넣어주었습니다. 그리고 nice값들은 math library 사용을 하지 않기 위해, 미리 계산하여 크기

가 5인 실수형 배열 nice에 넣어주었습니다. 전역변수로 선언했던 node 구조체의 포인터 변수인 head는 선언만 하고 메모리를 할당해주지 않았기 때문에, malloc함수로 node 크기만큼 메모리를 할당해주었습니다.

그 후에는 SIGALRM에 대한 signal handler를 설정하기 위해, sigaction 구조체의 변수인 sact에 sa_handler로 catcher 함수를 설정하여주었습니다. 그리고 timer를 세팅해주었습니다. 먼저 which를 이용해서 ITIMER의 세 가지 중, ITIMER_REAL를 사용하도록 하였고, itimerval 구조체 변수인 value를 이용하여 5초 후에 시작, 1초마다 timer가 울리게 미리 선언을 해놓았습니다.

그 다음에는 프로세스를 만들고 이를 노드에 넣기 위해 필요한 변수들을 선언해주었습니 문자형 변수 c는 'A'로 값을 할당하였는데, 프로세스마다, 즉 Node마다 Node->c값을 다르게 가지도록 구현할 때 사용하였습니다. 그리고 문자형 포인터 변수인 cin은 각 프로세스마다, 즉, 각 Node마다 다르게 할당된 알파벳을 받아서 execl에 parameter로 사용하기 위해 선언하였습니다. 알파벳 하나만 들어갈 것이기 때문에 malloc을 이용해 char 크기만큼 할당해주었습니다. Node 구조체 포인터 변수인 new도 malloc을 이용해 node 크기만큼 할당을 해주었는데 이는 새로운 자식 프로세스가 추가 될 때 마다, 새로운 노드를 만들 때 사용하는 Node입니다. 그리고 마지막으로 정수형 변수 pid를 선언해주었는데, 이는 fork를 통해 자식 프로세스를 생성할 때, fork에 대한 리턴값인 자식 프로세스의 pid를 담기 위한 변수입니다.

Nice 값마다 생성해야 할 프로세스 개수가 담긴 n의 크기가 5이기 때문에 첫 번째 for 문은 다섯 번 실행되도록 구현하였습니다. N[i] 값이 0이라면 아무것도 실행을 하지 않고, 0이 아니라면 두 번째 포문을 실행하게 되는데, 이 때 두 번째 for문은 n[i]의 값 만큼 실행이 되는데, n[i]의 값 만큼 프로세스를 만들기 위함입니다. 이 for문 안에서는 먼저 fork를 하고 이에 대한 return 값, 즉, 방금 생성된 자식 프로세스의 pid를 pid 변수에 담게됩니다. 그리고, 부모 프로세스라면, pid를 value로 갖고있는 Node를 createNode 함수를 이용해서 만들고 이를 new에 할당하게 됩니다. 그리고 new의 nice 값과 c의 값을 할당한 후, 모든 프로세스가 다른 c값을 갖게 하기 위하여 c의 값을 증가히켜 줍니다. 그리고 Linekd List에 new를 추가하기 위해서 insert함수의 parameter로 new를 전달하게 됩니다. 만약 자식 프로세스라면 execl 때 사용 할 cin 포인터 변수의 값으로 c를 할당하고 이중 for문을 벗어나게 합니다.

생성된 자식프로세스는 이중 for문을 벗어난 뒤, execl의 parameter로 cin을 전달하게 되고, ku_app의 kill(getpid(), SIGSTOP)에 의해 중지됩니다.

모든 프로세스를 생성하고, 부모 프로세스는 아까 선언하고 값을 할당해 놓은 which와 value를 setitimer의 parameter로 전달하여 타이머를 시작하게 됩니다. 그리고 time count가 time_slices보다 커질 때 까지, 즉 정해진 time slice만큼 실행이 될 때 까지 while문을 계속 돌게 만듭니다. 이 while문을 도는 동안, setitimer로 시작한 타이머가 SIGALRM을 발생시키고, 이 SIGALRM이 발생하자, 아까 signal handler로 할당한 catcher 함수에 의해 프로세스 스케줄링이 일어나게 됩니다.

이 프로세스 스케줄링에 의해 SIGCONT 를 받은 자식 프로세스는 아까 중단한 ku_app을 이어서 실행하게 됩니다.

지정된 time slice만큼 SIGALRM이 발생 하였다면, value의 모든 값을 0으로 세팅하고 다시 한번 setitimer를 통해 타이머를 발생시켜, 타이머를 멈춥니다. 그리고 아까 malloc으로 메모리를 할당한 new, cin, head의 메모리를 free해줍니다. 마지막으로, 생성한 모든 자식 프로세스들을 확실하게 kill하기 위해 SIGKILL을 보내주었습니다.

<Function>

createNode	Functionality	입력된 parameter 값을 value로 갖는 새로운 Node를 만들고, 이를 리턴하는 함수
	Parameters	Int
	Return Value	Node*

addFirst	Functionality	LinkedList의 맨 앞에, 즉, head 다음 노드로 입력된 Node를 넣는 함수
	Parameters	Node*
	Return Value	

insert	Functionality	Linked List에 있는 Node의 vrun 값과 입력된 Node의 vrun 값의 크기를 비교하여 작은 순서대로 배열될 수 있게 insert 하는 함수
	Parameters	Node*
	Return Value	

dequeue	Functionality	Vrun이 작은 순서대로 정렬되어 있는 Linked List에서 맨 앞의 Node, 즉, vrun이 가장 작은 Node를 삭제하고, 이를 리턴하는 함수
	Parameters	
	Return Value	Node*

catcher	Functionality	SIGALRM의 signal handler이다. SIGALRM 발생 시, 방금 실행된 프로세스의 vrun타임을 계산해서 이를 갱신해주고, insert 함수를 이용해서 해당 Node를 Linked List에 넣고, 다음으로 실행할 Node, 즉, vrun이 가장 작은 Node에게 dequeue 함수를 이용해서 SIGCONT를 보낸다.
	Parameters	int
	Return Value	