

Mini OS 제작

(Thread 및 Scheduling 구현)

숭실대학교
전자정보공학부 IT 융합전공

20192579 김용준
20192594 박태성
20192617 이예진
20192606 정승준

목차

1. 서론 -----	3
1. 1 프로젝트 목표	
1. 2 주차 별 진행 내용	
1. 3 역할 분배	
2. 본론 -----	5
2. 1 Threading	
2. 1 1구현 내용	
2. 1 2코드 설명	
2. 1 3 문제 및 해결	
2. 2 Scheduling	
2. 2. 1구현 내용	
2. 2. 2코드 설명	
2. 2. 3 문제 및 해결	
2. 3 병합 및 메모리, UI	
2. 3. 1구현 내용	
2. 3. 2코드 설명	
2. 3. 3 문제 및 해결	
3. 결론 -----	10
3. 1 Threading	
3. 2 Scheduling	
3. 3 병합 및 메모리 UI	
3. 4 결론	
4. 참고자료 -----	12

1. 서론

1. 1 프로젝트 목표

User 모드에서 명령을 실행하면 kernel에서 처리하는 작업을 직접 설계하는 Mini OS 제작

Process Management 중 Thread 및 Scheduling 구현

1. 2 주차 별 진행 내용

7주차

정기 회의 날짜 결정 및 프로젝트 고려사항 토의

8주차

프로세스 관리 및 동기화 복습

9주차

프로세스 관리 학습 및 Mini OS 관련 github 자료 분석

10주차

프로세스 및 스레드, 메모리 관련 학습

Mini OS 관련 github 자료 분석

역할 분배

11주차

Multi-thread 관련 자료 분석

Scheduling 코드 작성 및 실행

12주차

Thread.c 코드 작성 및 실행

Scheduling 코드 최적화

13주차

Multi-thread 관련 자료 분석

Scheduling 중 interrupt 처리에 대해 코드 구현

코드 병합

14주차

최종 프로젝트 구현

발표준비 및 보고서 작성

1. 3 역할 분담

인원	역할
김용준	Process 관리 및 메모리 관련 자료조사, Scheduling 작업진행, 코드 병합 및 UI, 메모리 구현
박태성	Process 및 thread에 대한 자료 조사, thread 및 Multi-thread 작업 진행
이예진	Mini OS 관련 github 자료 분석, thread 및 Multi-thread 작업 진행
정승준	Mini OS 관련 github 자료 분석, Scheduling 작업진행, Scheduling 추가 구현
공통	보고서 및 발표자료 제작

2. 본론

2. 1 Threading

2. 1. 1 구현 내용

- A. pthread라이브러리를 참고하여 자체 thread 라이브러리인 thread.h를 생성하여, thread 역할을 하는 my_thread 구조체를 구현한다.
- B. pthread 라이브러리의 pthread_create와 pthread 라이브러리의 pthread_exit함수 역할을 하는 thread_wrapper 함수를 구현한다.
- C. 두 thread간의 context switching을 담당하는 schedule 함수를 구현한다.

2. 1. 2 코드 설명

- A. 멤버 변수로 thread의 start_routine함수와 그 매개변수인 arg 포인터, 스택 포인터용 stack포인터와 다음 thread를 가리키는 next의 my_thread 구조체 포인터, 마지막으로 context switch용 ucontext_t context변수의 my_thread 구조체를 생성한다.
- B. pthread_create 함수를 참고하여 my_thread구조체의 멤버 변수들의 초기화를 진행하며, 이 과정에서 context 멤버를 초기화 할 때 getcontext 함수를 사용하고, my_thread를 thread_queue에 연결함
- C. schedule함수의 작동은 현재 thread와 thread_queue의 thread를 context switching 하는 과정을 ucontext 라이브러리의 swapcontext 함수를 사용한다.
- D. my_thread 구조체의 멤버인 start_routine 함수를 thread_wrapper를 통해 arg 멤버 변수를 전달해 주며 실행하고, 현재 thread의 stack의 메모리 할당을 해제한 뒤 현재 thread를 NULL로 설정하고 schedule 함수를 시행한다.

2. 1. 3 문제 및 해결

- A. 내부 context switching 기능을 구현하면서 jmp_buf와 longjmp를 사용하여 구현하려고 했지만, 버스 오류가 발생하여 프로그램이 종료되는 문제가 발생했다.
 - 1) Memory에 대한 버스 오류가 원인임을 확인했다.
 - 2) ucontext 라이브러리를 사용하여 이 문제를 swapcontext 함수를 사용하여 해결한다.
- B. 부모 프로세스에서 thread1과 thread2를 생성하고, printf를 반복하는 thread 함수를 실행한 뒤 schedule 함수를 이용해 두 thread간의 context switching 실행할 시 버스 오류가 발생했다.
 - 1) 해당 부분은 프로젝트에 큰 부분이 없을 것으로 보여 삭제했다.

2. 2 Scheduling

2. 2. 1 구현 내용

- A. Weighted round robin 구현
 - 1) 기존 RR 알고리즘에서 서로 다른 프로세스나 작업에 서로 다른 실행 우선순위를 부여하는 방식으로 스케줄링 구현한다.
 - 2) Weighted round robin을 위한 Thread_Processing, addProcessToQueue, customScheduling을 구현한다.
- B. 이점
 - 1) 프로세스나 작업 간의 서로 다른 실행 우선순위를 부여할 수 있는 점에 RR 방식에 비해 특정 프로세스(스레드)에 더 많은 CPU를 할당할 수 있다.
 - 2) 더욱 복잡한 알고리즘(SJF, MLFQ)에 비해 오버헤드가 낮아 유지 및 관리가 용이하다.
 - 3) 예측 가능한 동작을 제공할 수 있어 일관된 성능이 필수인 실시간 시스템과 애플리케이션에 적합하다.
- C. Aging 구현(UpdatePriority 구현)
 - 1) Starvation을 고려하여 순위가 낮은 스레드를 위로 올린다.

2. 2. 2 코드 설명

- A. Weighted Round Robin을 구현하기 위해 Queue 자료구조를 이용한다.(High_queue, Medium_queue, Low_queue)
- B. 각 스레드의 우선순위를 저장하는 변수 지정-> 해당 우선 순위에 따라 각 queue에 enqueue를 실행한다.
- C. 해당 큐에 따른 가중치를 다르게 분배한다.
- D. 큐에 있는 스레드 개수에 가중치를 곱한 값들을 저장한 후 total_weighted를 이용해 총 합 저장한다.
- E. $random = rand() \% total_weighted$ 을 하는 샘플링을 한 후 해당 수에 따라 큐에 진입한다.(가중치 및 count에 따라 각 queue에 진입을 조절 할 수 있다.)
- F. 각 queue에 있는 스레드를 주어진 가중치만큼 처리를 하며, 한 스레드를 처리했을 때 가중치가 남은 경우 해당 queue의 다음 스레드로 넘어가서 처리한다.
- G. medium_queue, low_queue에서 wait_time이 일정 수준을 넘을 경우 time에 따라 위의 순위에 enqueue처리한다.

2. 2. 3 문제 및 해결

- A. 해당 스레드가 I/O 처리가 필요한 경우를 고려한다.
 - 1) updateWaitQueues함수 및 wait_queue를 선언하고 interrupt 를 처리한다.
 - 2) Queue.h 파일에 interrupt_Waitingtime이라는 함수 정의를 통해 기존 scheduling큐들의 스레드 처리 시간과 I/O처리 시간이 동일하게 흐를 수 있게 한다.

2-3 코드 병합 및 메모리, UI

2. 3. 1 구현 내용

A. 기존 코드 통합

- 1) threading 코드 및 scheduling 코드를 통합하여 구현한다.
- 2) Scheduling에 필요한 자료구조를 추가한다.

B. 메모리 관리

- 1) 프로세스가 생성되고 실행되는 과정에서 디멘드 페이징이 일어나는 과정을 최대한 단순화한다.
- 2) 해당 프로세스가 어떤 페이지들을 가지고 있으며, 어떤 물리 프레임에 매핑되어있는지, 그리고 페이지 폴트 발생 여부와 페이지를 디스크에서 다시 가져오는 것을 시뮬레이션한다.
- 3) 페이지 교체 알고리즘(LRU) 또한 모사하였다.

C. UI 구현

- 1) 인터럽트 등의 특정 상황을 표현하기 위해 Process id에 색을 입혀 인터럽트 되어 대기 큐에 들어가 있는 상황을 표현했다.

2. 3. 2 코드 설명

- A. threading 및 scheduling은 이전 코드와 동일하며 기존 scheduling에 필요한 자료구조를 추가한다.
- B. UI 구현은 ANSI 이스케이프 문자를 활용해 터미널의 문자를 지우고 다시 출력하는 식으로 제작했다.
- C. 프로세스 스케줄링 알고리즘의 평가 척도인 CPU 이용률, 턴어라운드 시간 등을 stats라는 코드를 이용하여 스케줄링이 진행될 때 기록하였다.

2. 3. 3 문제 및 해결

- A. 코드를 병합하는 과정에서 기존 스레드 자료구조를 스케줄링할 때 커스터마이징한 코드와 계속해서 호환이 되지않는 문제가 발생하였다.

- 1) 스레드 자료 구조에는 ucontext와 thread_function 코드를 사용했으나 자료구조 숙련도 문제로 기존 스레드 코드에서 사용한 ucontext와 thread_function은 제거하고, 프로세스 자료구조로 전환한 다음 스레드 자료 구조를 넣어 최대한 단순하게 만들어서 포함시켰다.
- 2) Thread_function 코드를 전달하여 시뮬레이션하는 과정에서 오류가 많이 발생하여, UI 기반으로 스케줄링 되고 있음을 스케줄러 입장에서 구현하도록 했다.

3. 결론

3. 1 Threading

```
kriek@kriek-virtual-machine: ~  
kriek@kriek-virtual-machine:~$ vim m.c  
/usr/share/vim/vimrc 수행중 에러 발견:  
69 줄:  
E518: 모르는 옵션: euc-kr  
계속하려면 엔터 혹은 명령을 입력하십시오  
kriek@kriek-virtual-machine:~$ gcc -o m m.c  
kriek@kriek-virtual-machine:~$ ./m  
Main thread starting  
Thread 1: started  
Thread 1: 0  
Thread 1: 1  
Thread 1: 2  
Thread 1: 3  
Thread 1: 4  
Thread 1: finished  
Thread 2: started  
Thread 2: 0  
Thread 2: 1  
Thread 2: 2  
Thread 2: 3  
Thread 2: 4  
Thread 2: finished  
kriek@kriek-virtual-machine:~$
```

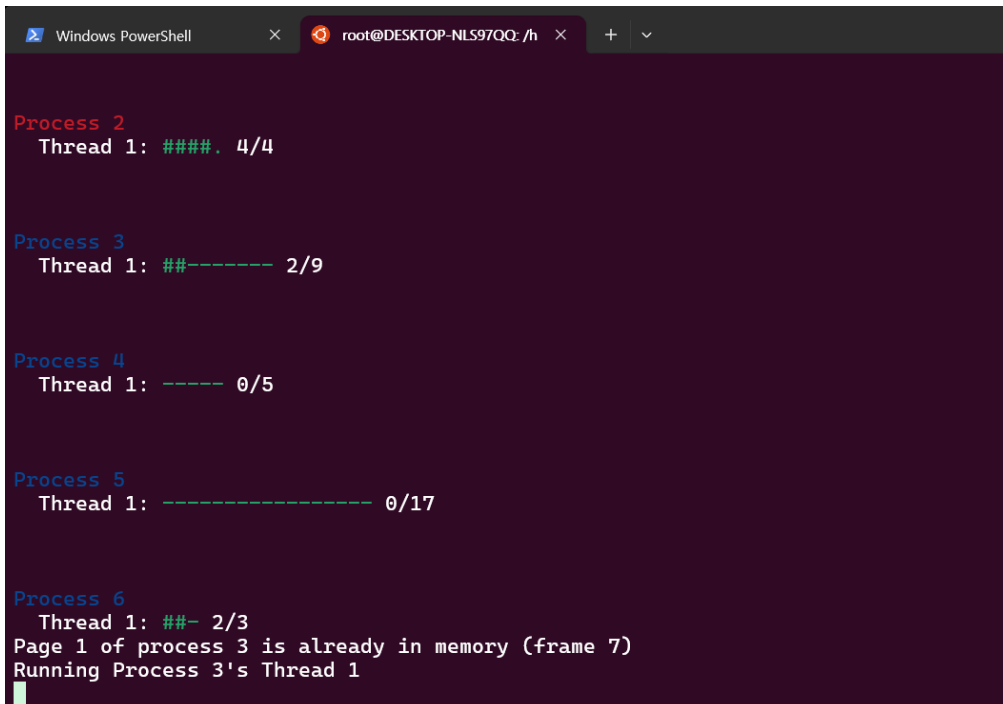
스레드를 생성하고 처리하는 과정이 잘 나타나고 있다.

3. 2 Scheduling

```
초기 상태  
Queue elements: 1 4 7 10  
Queue elements: 2 5 8  
Queue elements: 3 6 9  
  
high접근  
Thread 1 is pushed in wait_queue  
  
Thread 4 from queue, and its waiting time is 0.  
rest_weight: 3  
  
Wait Queue(id, waiting_time, interrupt_time): (1, 0, 6)  
현재 남은 프로세스 총 갯수: 8  
  
Thread 7 is pushed in wait_queue  
  
Thread 10 from queue, and its waiting time is 4, and still continue  
value current remaining: 7
```

스케줄링이 잘 처리되고 기록이 되고 있다.

3. 3 코드 병합 및 메모리, UI



```
Windows PowerShell
root@DESKTOP-NLS97QQ: /h

Process 2
Thread 1: ####. 4/4

Process 3
Thread 1: ##----- 2/9

Process 4
Thread 1: ----- 0/5

Process 5
Thread 1: ----- 0/17

Process 6
Thread 1: ##- 2/3
Page 1 of process 3 is already in memory (frame 7)
Running Process 3's Thread 1
```

프로세스 및 스레드에 관한 정보가 출력이 되고 있으며 각 스레드의 진행상황을 UI를 통해 출력되고 있다.

3. 4 결론

라이브러리를 모방한 thread.h 파일을 제작하고 기존 스케줄링 알고리즘과는 다른 방식의 스케줄링을 구현했다. 구현 과정 중에 버스 오류가 나거나 다른 상황에서 고려를 해야 했으며 이를 해결하는 과정이 필요했다. 또한 제작한 구현을 통합하는 과정에서 추가적인 문제가 있었으며, 자료구조 전환 및 스케줄러 입장 구현 등으로 수행했다. 이를 통해 현재 MiniOS 기능을 구현하고 실제 시연을 할 수 있다. 물론 실제 OS는 파일 시스템 등의 더 많은 기능을 수행하고 있으며 이에 대한 추가적인 헤더파일 및 C파일을 구현해야 한다. 이는 앞으로 더 프로젝트를 진행한다면 해야 할 과제라고 생각한다. 현재는 스레딩 및 스케줄링을 할 수 있으며 이를 모니터링 하여 UI를 통해 보여주는 것으로 이 프로젝트를 마친다.

4. 참고자료

Abraham Silberschatz. (2020), Operating System Concepts. (10th ed.).퍼스트북.