

과제 #3 : xv6에서 시스템 호출 추가 및 스케줄러 수정

○ 과제 목표

- xv6의 프로세스 관리와 스케줄링 기법 이해
- xv6에서 시스템 호출 추가 및 기존 스케줄러 변경

○ 기본 지식

- 시스템 호출 추가 방법 이해
 - ✓ 기존 시스템 호출의 구현을 따라 새 시스템 호출을 추가하는 방법을 이해
 - 특정 시스템 호출은 인자가 없고 정수값만 리턴
 - ☞ 예 : sysproc.c 내 구현된 uptime()
 - 특정부 시스템 호출은 문자열 및 정수 등 여러 인수를 받아 간단한 정수 값을 리턴
 - ☞ 예 : sysfile.c 내 구현된 open()
 - 특정 시스템 호출은 여러 정보를 사용자가 정의한 구조체로 사용자 프로그램에 리턴
 - ☞ 예. fstat()은 파일에 대한 정보를 struct stat를 넣고 이 구조체를 가져와서 ls 응용 프로그램에 의해 파일에 대한 정보를 표준 출력
- Cross Compile 방법 학습
 - ✓ xv6에는 텍스트 편집기 또는 gcc 컴파일러가 없음. 따라서 자신의 리눅스 시스템에서 vi를 이용하여 프로그램 작성하고 컴파일하고 나온 실행파일을 xv6 상에서 수행
- xv6 커널 이해
 - ✓ proc.c, proc.h, syscall.c, syscall.h, sysproc.c, user.h, usys.S 수정 필요
 - user.h : xv6의 시스템 호출 정의
 - usys.S : xv6의 시스템 호출 리스트
 - syscall.h : 시스템 호출 번호 매핑. -> 새 시스템 호출을 위해 새로운 매핑 추가
 - syscall.c : 시스템 호출 인수를 구문 분석하는 함수 및 실제 시스템 호출 구현에 대한 포인터
 - sysproc.c : 프로세스 관련 시스템 호출 구현. -> 여기에 시스템 호출 코드를 추가
 - proc.h는 struct proc 구조 정의 -> 프로세스에 대한 추가 정보를 추적을 위해 구조 변경
 - proc.c : 프로세스 간의 스케줄링 및 컨텍스트 전환을 수행하는 함수

○ 과제 내용

- 0. xv6 및 qemu 다운로드 및 설치 (별도 보고서 쓰지 않아도 됨)
 - ✓ 주어진 xv6(xv6-modified.tgz)를 설치 후 실행 (ubuntu 18.04 사용 권장)
 - ✓ 구글 클래스룸에서 다운로드 가능. (주의, 다른 사이트 것 다운 받으면 절대 안됨. 이미 일부 내용을 수정한 상태이기 때문에 본 구글 클래스룸에서만 다운 받아 사용할 것)

```

oslab@oslab-VirtualBox: ~/workspace/xv6-public
File Edit View Search Terminal Help
oslab@oslab-VirtualBox:~/workspace/xv6-public$ make qemu
qemu-system-i386 -serial mon:stdio -drive file=fs.img,index=1,media=disk,format=
raw -drive file=xv6.img,index=0,media=disk,format=raw -smp 1 -m 512
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
$

```

[그림 1] make qemu 실행 결과

```

oslab@oslab-VirtualBox: ~/workspace/xv6-public
File Edit View Search Terminal Help
cat                2 3 13856
echo               2 4 12864
forktest          2 5 8300
grep              2 6 15732
init              2 7 13448
kill              2 8 12920
ln                2 9 12816
ls                2 10 15000
mkdir             2 11 13000
rm                2 12 12976
sh                2 13 23464
stressfs          2 14 13648
usertests         2 15 56576
wc                2 16 14396
zombie            2 17 12640
hello_test        2 18 12608
helloname_test    2 19 12632
getnumproc_test   2 20 12704
getmaxpid_test    2 21 12676
getprocinfo_test  2 22 13192
seqinc_prio       2 23 15696
seqdec_prio       2 24 15680
console           3 25 0
$ █

```

[그림 2] ls 실행 결과

- ✓ qemu 설치
 - \$ make -> \$ make qemu
 - xv6 운영체제는 자신의 컴퓨터에서 x86 하드웨어를 에뮬레이트하는 QEMU라는 x86 에뮬레이터에서 실행됨 (에뮬레이터가 없어도 운용 가능하나, 수정을 위해서 에뮬레이터를 사용을 권장)
 - 구글 클래스룸에서 다운로드 가능. (주의, 아래 사이트 등에서 직접 받아도 됨)
 - ☞ <https://github.com/qemu/qemu>
 - ☞ apt-get install qemu-kvm
- 1. 새로운 시스템 호출 추가 및 간단한 응용 프로그램 구현
 - ✓ (1) hello() 시스템 호출 추가 및 이를 호출하는 주어진 간단한 응용 프로그램 실행 (테스트 1)
 - helloxv6를 콘솔에 출력하는 hello() 시스템 호출 구현
 - 커널 모드의 cprintf() 사용

- 주어진 응용 프로그램(hello_test.c, xv6-modified.tgz 내 포함되어 있음)을 실행시키고 결과 화면 캡처

```

oslab@oslab-VirtualBox: ~/workspace/xv6-public
File Edit View Search Terminal Help
forktest          2  5  8300
grep              2  6 15732
init              2  7 13448
kill              2  8 12920
ln                2  9 12816
ls               2 10 15000
mkdir            2 11 13000
rm               2 12 12976
sh               2 13 23464
stressfs         2 14 13648
usertests        2 15 56576
wc               2 16 14396
zombie           2 17 12640
hello_test       2 18 12608
helloname_test   2 19 12792
getnumproc_test  2 20 12704
getmaxpid_test   2 21 12676
getprocinfo_test 2 22 13192
seqinc_prio      2 23 15696
seqdec_prio      2 24 15680
console          3 25  0
$ hello_test
helloxv6
s

```

[그림 3] hello_test 실행 결과 예

- ✓ (2) hello_name() 시스템 호출 추가 및 이를 호출하는 간단한 응용 프로그램 구현 (테스트 2)
 - 문자열 이름을 콘솔에 출력하는 helloname(name)시스템 호출 구현
 - 커널 모드의 cprintf() 사용
 - 간단한 응용 프로그램(helloname_test.c)을 위 (1)의 주어진 응용 프로그램과 비슷한 응용 프로그램 작성하고 이를 실행 시킨 결과 화면 캡처

```

oslab@oslab-VirtualBox: ~/workspace/xv6-public
File Edit View Search Terminal Help
init              2  7 13448
kill              2  8 12920
ln                2  9 12816
ls               2 10 15000
mkdir            2 11 13000
rm               2 12 12976
sh               2 13 23464
stressfs         2 14 13648
usertests        2 15 56576
wc               2 16 14396
zombie           2 17 12640
hello_test       2 18 12608
helloname_test   2 19 12792
getnumproc_test  2 20 12704
getmaxpid_test   2 21 12676
getprocinfo_test 2 22 13192
seqinc_prio      2 23 15696
seqdec_prio      2 24 15680
console          3 25  0
$ hello_test
helloxv6
$ helloname_test xv6
hello xv6
$

```

[그림 4] helloname_test 실행 결과 예

- ✓ (3) get_num_proc() 시스템 호출 추가 및 이를 호출하는 간단한 응용 프로그램 구현 (테스트 3)
 - 시스템에서 활성화되어 있는 프로세스 수 리턴(embryo상태, running상태, runnable상태, sleeping 상태, zombie 상태 등 모든 상태의 프로세스 포함)하는 get_num_proc() 시스템 호출 구현
 - 간단한 응용 프로그램(getnumproc_test.c)을 위 (1)의 주어진 응용 프로그램과 비슷한 응용 프로그램 작성하고 이를 실행 시킨 결과 화면 캡처

```

oslab@oslab-VirtualBox: ~/workspace/xv6-public
File Edit View Search Terminal Help
ln                2  9 12816
ls                2 10 15000
mkdir            2 11 13000
rm               2 12 12976
sh               2 13 23464
stressfs         2 14 13648
usertests        2 15 56576
wc               2 16 14396
zombie           2 17 12640
hello_test       2 18 12608
helloname_test   2 19 12792
getnumproc_test  2 20 12704
getmaxpid_test   2 21 12676
getprocinfo_test 2 22 13192
seqinc_prio      2 23 15696
seqdec_prio      2 24 15680
console          3 25  0
$ hello_test
helloxv6
$ helloname_test xv6
hello xv6
$ getnumproc_test
Total Number of Active Processes: 3
$

```

[그림 5] getnumproc_test 실행 결과 예

- ✓ (4) get_max_pid() 시스템 호출 추가 및 이를 호출하는 간단한 응용 프로그램 구현 (테스트 4)
 - 시스템에서 활성화되어 있는 모든 프로세스의 PID 중 최대 PID (즉, 프로세스 테이블에서 슬롯을 차지함)를 리턴하는 get_max_pid() 시스템 호출 구현
 - 간단한 응용 프로그램(getmaxpid_test.c)을 위 (1)의 주어진 응용 프로그램과 비슷한 응용 프로그램 작성하고 이를 실행 시킨 결과 화면 캡처

```

oslab@oslab-VirtualBox: ~/workspace/xv6-public
File Edit View Search Terminal Help
mkdir            2 11 13000
rm               2 12 12976
sh               2 13 23464
stressfs         2 14 13648
usertests        2 15 56576
wc               2 16 14396
zombie           2 17 12640
hello_test       2 18 12608
helloname_test   2 19 12792
getnumproc_test  2 20 12704
getmaxpid_test   2 21 12676
getprocinfo_test 2 22 13192
seqinc_prio      2 23 15696
seqdec_prio      2 24 15680
console          3 25  0
$ hello_test
helloxv6
$ helloname_test xv6
hello xv6
$ getnumproc_test
Total Number of Active Processes: 3
$ getmaxpid_test
Maximum PID: 7
$

```

[그림 6] getmaxpid_test 실행 결과 예

- ✓ (5) get_proc_info(pid, *processInfo) 시스템 호출 추가 및 간단한 응용 프로그램 구현 (테스트 5)
 - 정수 PID와 processInfo 구조체에 대한 포인터를 인자로 받아 사용자 모드와 커널 모드 간 정보를 전달하는 데 사용하는 get_proc_info(pid, & processInfo) 시스템 호출 추가 및 이를 호출하는 간단한 응용 프로그램 구현
 - processInfo.h 참고, user.h와 proc.c에 processInfo.h 포함 등 수정
 - processInfo 구조체의 멤버 변수 추가 및 출력 프로그램 작성
 - 리턴되어야 하는 프로세스에 대한 정보는 부모 PID 및 스케줄러에 의해 문맥교환(context switch) 횟수와 프로세스 크기(byte 단위로)가 포함되어야 함

- 프로세스에 대한 정보는 중 일부는 프로세스의 struct proc에서 추출 가능. 그러나 기존에 없는 추가 정보를 추출하려면 새로운 맴버 변수(필드)를 추가해야 함
- 저장된 PID를 가진 프로세스가 없는 경우 에러로 -1을 리턴
- 간단한 응용 프로그램(getprocinfo_test.c)을 위 (1)의 주어진 응용 프로그램과 비슷한 응용 프로그램 작성하고 이를 실행 시킨 결과 화면 캡처

```

oslab@oslab-VirtualBox: ~/workspace/xv6-public
File Edit View Search Terminal Help
wc                2 16 14396
zombie            2 17 12640
hello_test        2 18 12608
helloname_test    2 19 12792
getnumproc_test   2 20 12704
getmaxpid_test    2 21 12676
getprocinfo_test  2 22 13192
seqinc_prio       2 23 15696
seqdec_prio       2 24 15680
console           3 25 0
$ hello_test
helloxv6
$ helloname_test xv6
hello xv6
$ getnumproc_test
Total Number of Active Processes: 3
$ getmaxpid_test
Maximum PID: 7
$ getprocinfo_test
PID    PPID    SIZE    Number of Context Switch
1       0      12288    26
2       1      16384    32
8       2      12288     8
$
  
```

[그림 7] getprocinfo_test 실행 결과 예

- 2. 새로운 스케줄러를 위한 2개의 시스템 호출 추가
 - ✓ 프로세스의 우선순위를 고려한 스케줄러 설계 및 기존 xv6 스케줄러를 수정
 - ✓ 새로운 시스템 호출(아래 (1), (2))을 추가하여 프로세스 우선순위 설정(set) 및 가져오기(get)
 - ✓ (1) set_prio() 및 (2) get_prio() 시스템 호출 추가 및 이를 호출하는 간단한 응용 프로그램 구현
 - 프로세스가 setprio(n) 시스템 호출하면 프로세스의 우선순위는 n으로 설정
 - n은 양의 정수이며, 클수록 더 높은 우선 순위
 - 프로세스 proc 구조체 내 우선순위를 설정하고 가져오는 기능 포함
 - 성공 시 0을 리턴, 실패 시 음수 값을 리턴
- 프로세스 테이블 구조체인 ptable은 lock으로 보호함을 주의 : ptable을 읽기/쓰기를 위해 ptable에 접근하기 전에 lock을 획득해야 함. 읽기/쓰기를 완료하면 lock을 해제해야 함. 미묘한 버그를 피할 수 있게 lock을 잘 써야 함.
- 3. 우선순위 기반 RR 스케줄러 구현
 - ✓ xv6의 현재 스케줄러는 기본 라운드-로빈(RR)스케줄러임
 - ✓ 사용자가 정의한 프로세스 우선순위가 적용된 스케줄러를 구현하기 위해 기존 RR 스케줄러 수정
- 우선순위 기반 RR 스케줄러 구현하기 위해 우선순위를 사용하여 다음 번에 수행할 프로세스 선택
 - ✓ (a) 우선순위가 높을수록 프로세스의 CPU 시간이 더 많아져서 우선순위가 높은 프로세스가 우선 순위가 낮은 프로세스보다 빨리 완료
 - ✓ (b) 우선순위가 낮은 프로세스가 너무 부족하지 않아야 하고, 우선순위가 더 높은 프로세스는 CPU time quantum을 약간 더 가져야 함
 - ✓ 우선순위가 설정(변경)되기 전에 프로세스의 기본 우선순위는 정해야 함.
 - ✓ 커널 데이터 구조체에 접근 시 lock 이용해야 함

- 4. 새로운 스케줄러를 테스트 및 검증

- ✓ 주어진 스케줄링 테스트 프로그램(seqinc_prio.c 와 secdec_prio.c, xv6-modified.tgz 내 포함되어 있음) 컴파일하고 셸에서 이를 실행하여 결과 출력. 이를 통해 구현한 3의 우선순위 스케줄러의 기능 검증 (테스트 6, 테스트 7)
- ✓ 만약 주어진 스케줄링 테스트 프로그램(seqinc_prio.c 와 secdec_prio.c)이 수정한 스케줄러와 잘 맞지 않아 예상하지 못한 출력을 할 경우, 스스로 스케줄링 테스트 프로그램을 별도로 만들고 검증 (이 경우 별도 소스코드 제출해야 함, 파일명은 자유로 지정)

```
oslab@oslab-VirtualBox: ~/workspace/xv6-public
File Edit View Search Terminal Help
$ getmaxpid_test
Maximum PID: 7
$ getproinfo_test
PID  PPID  SIZE  Number of Context Switch
1      0    12288    15
2      1   16384    30
8      2   12288    7
$ seqinc_prio 5
Priority of parent process = 1000
All children completed
Child 0.  pid 10
Child 1.  pid 11
Child 2.  pid 12
Child 3.  pid 13
Child 4.  pid 14
Exit order
pid 14
pid 13
pid 12
pid 11
pid 10
$
```

[그림 8] seqinc_prio 실행 결과 예

```
oslab@oslab-VirtualBox: ~/workspace/xv6-public
File Edit View Search Terminal Help
Exit order
pid 14
pid 13
pid 12
pid 11
pid 10
$ seqdec_prio 5
Priority of parent process = 1000
All children completed
Child 0.  pid 16
Child 1.  pid 17
Child 2.  pid 18
Child 3.  pid 19
Child 4.  pid 20
Exit order
pid 16
pid 17
pid 18
pid 19
pid 20
$
```

[그림 9] seqdec_prio 실행 결과 예

○ 과제 제출 마감

- 2020년 10월 14일 (수) 23시 59분 59초까지 구글클래스룸(이전한 곳, <https://classroom.google.com/c/MTU0NTQ2MTc0MzA2?cjc=qgprwwb>)으로 제출
- 보고서 (hwp, doc, docx 등으로 작성 - 총 7개의 테스트 프로그램이 수행된 결과 캡처 등이 포함된) 와 주어진 소스코드에서 변경한 소스코드 (변경된 부분, 단 주어진 코드 이름과 같게), 주어진 소스코드, 테스트 프로그램 총 4개 (테스트 2, 3, 4, 5는 제출해야 함. 또한 별도로 학생이 구현한 테스트 프로그램 있으면 함께 제출. 단, 주어진 테스트 프로그램, 테스트 1, 6, 7은 제출 불필요.)
- 1일 지연 제출마다 30% 감점. 4일 지연 제출 시 0점 처리 (이하 모든 설계 과제 동일하게 적용)

○ 필수 구현(설치 및 설명 등)

- 1, 2, 3
 - ✓ 3은 4의 테스트 프로그램에서 일부 오류가 나오는 것까지 실제 RR을 구현한 것을 의미
 - ✓ 3의 오류가 발생하는 상황을 이미 확인하였으며, 학생들이 오류를 수정하지 못한 상태이면 4의 테스트 프로그램이 제대로 실행되지 않음. 이 경우 3은 완료한 것으로 하고 4는 완료하지 못한 것으로 처리 예정

○ 배점 기준

- 1-(1) : 2점 / 1-(2) : 3점 / 1-(3) : 5점 / 1-(4) : 5점 / 1-(5) : 5점
- 2 : 30점
- 3 : 40점
- 4: 10점