Operating Systems
Project 02

- Advanced Process Management -

Due date : 11:59 pm June 07

프로젝트 목표

- xv6의 프로세스는 일반적인 운영체제에서 필요한 최소한의 기능들만을 가지고 있습니다.
- 기본 구조에 보다 다양한 기능들, 특히 메모리 관련 요소들을 추가하고 프로세스들을 쉽게 관리할 수 있는 도구를 만들어 xv6를 더욱 풍부하게 만들어 봅시다.

구현할 기능들

- Administrator mode
- Custom stack size
- Per-process memory limit
- Shared memory
- Process manager

Administrator Mode

- 프로세스의 실행 모드를 user mode / administrator mode로 구분하게 합니다.
- 프로세스가 실행되면 기본으로 user mode로 시작하며, 시스템 콜을 통해 관리자 권한을 획득할 수 있습니다.
- 관리자 권한을 획득한 프로세스는 이후 추가할 일부 시스템 콜을 사용할 수 있게 되며, 그 목록은 후술합니다.
- 구현할 시스템 콜
  - int getadmin(char *password);
  - 올바른 암호는 자신의 학번입니다.
  - password가 학번과 일치할 시 이 시스템 콜을 호출한 프로세스는 관리자 권한을 획득하고 getadmin 함수는 0을 반환합니다. 일치하지 않을 경우 -1을 반환합니다.

Custom Stack Size

- 기존의 xv6 프로세스는 exec 시에 하나의 스택용 페이지와 하나의 가드용 페이지를 할당받습니다.
- 스택용 페이지를 처음 실행 시에 원하는 개수만큼 할당받을 수 있게 하는 exec2 시스템 콜을 만듭니다.
  - int exec2(char *path, char **argv, int stacksize);
  - 첫 번째와 두 번째 인자의 의미는 기존 exec 시스템 콜의 첫 번째, 두 번째 인자와 동일합니다.
  - stacksize에는 스택용 페이지의 개수를 전달받습니다.
    - stacksize는 1 이상 100 이하의 정수여야 합니다.
    - 가드용 페이지는 이 크기에 관계없이 반드시 1개를 할당해야 합니다.

Custom Stack Size

- exec2 시스템 콜은 administrator mode에서만 호출할 수 있습니다.
- exec2 시스템 콜이 실패할 시 -1을 반환합니다.
  - user mode에서 호출한 경우, stacksize의 크기가 1 이상 100 이하의 정수가 아닌 경우 등에도 프로세스를 실행하지 않고 -1을 반환해야 합니다.
  - exec는 성공하면 반환하지 않으므로 exec2 시스템 콜 역시 성공 시의 반환값은 없습니다. (아무렇게나 지정해도 됩니다.)

Per-process Memory Limit

- 특정 프로세스에 대해 할당받을 수 있는 메모리의 최대치를 지정할 수 있게 합니다.
- 프로세스가 처음 생성될 때에는 제한이 없으며, 이후 시스템 콜을 통해 이 제한을 설정할 수 있습니다.
- 유저 프로세스가 메모리를 추가로 할당받으려 할 때 그 프로세스의 memory limit을 넘으려는 경우 변경하지 않도록 해야 합니다.
- 이 과제에서 프로세스가 사용한 메모리의 양은 프로세스 구조체의 sz의 크기로 합니다.

Per-process Memory Limit

- 구현할 시스템 콜
  - int setmemorylimit(int pid, int limit);
  - administrator mode에서만 호출할 수 있습니다.
  - pid는 memory limit을 지정할 프로세스의 pid입니다.
  - limit은 그 프로세스가 사용할 수 있는 memory limit을 바이트 단위로 나타냅니다.
    - limit의 값은 0 이상의 정수이며, 양수인 경우 그 크기만큼의 memory limit을 가지고, 0인 경우 제한이 없습니다.
  - 이미 그 프로세스가 할당받은 메모리보다 제한을 작게 설정하려는 경우 제한을 변경하지 않고 -1을 반환합니다.
  - pid가 존재하지 않는 경우, limit이 음수인 경우 등에도 -1을 반환합니다.
  - 정상적으로 동작했다면 0을 반환합니다.

Shared Memory

- 여러 프로세스가 동시에 접근할 수 있는 shared memory를 만듭니다.
- 각 프로세스가 생성될 때 그 프로세스 소유의 shared memory를 위한 페이지가 하나 할당됩니다.
- 이 페이지는 어떤 프로세스든 read는 가능하지만, write는 그 페이지의 소유 프로세스만 가능합니다.
  - 소유자가 아닌 프로세스가 해당 페이지에 write를 시도할 시 page fault exception이 발생해야 합니다.

Shared Memory

- 구현할 시스템 콜
  - char *getshmem(int pid);
  - pid가 소유한 shared memory 페이지의 시작 주소를 반환합니다.
- 프로세스가 종료될 때 shared memory 또한 올바르게 회수되어야 합니다.
- Shared memory는 프로세스 구조체의 sz 크기에 반영하지 않습니다.
  - tip: 다른 유저 페이지들과 분리하여 관리하고, 매핑은 커널 영역의 주소 그대로를 pgdir에 매핑하면 됩니다.

Process Manager

- 현재 실행 중인 프로세스들의 세부 정보를 확인하고 쉽게 관리할 수 있는 유저 프로그램을 만듭니다.
- 실행 후 종료 명령어가 들어올 때까지 한 줄씩 명령어를 입력받아 실행하는 방식으로 동작합니다.
- 프로그램의 이름은 pmanager로 합니다.
- pmanager는 시작할 때 getadmin 시스템 콜을 사용하여 관리자 권한을 획득해야 합니다.
- 실행 예시는 별도로 첨부한 동영상 파일을 참고하세요.

Process Manager

- pmanager는 다음과 같은 명령어들을 지원해야 합니다.
  - list
    - 현재 실행 중인 프로세스들의 정보를 출력합니다. 출력되어야 하는 것으로는 각 프로세스의 이름, pid, 실행 후 지난 시간, 스택용 페이지의 개수, 할당받은 메모리의 크기, 메모리의 최대 제한이 있습니다.
    - 프로세스의 정보를 얻기 위한 시스템 콜은 자유롭게 정의해도 됩니다.
  - kill <pid>
    - pid를 가진 프로세스를 kill합니다. kill 시스템 콜을 사용하면 됩니다.
    - 성공 여부를 출력합니다.

Process Manager

- pmanager는 다음과 같은 명령어들을 지원해야 합니다.
  - execute <path> <stacksize>
    - path의 경로에 위치한 프로그램을 stacksize 개수만큼의 스택용 페이지와 함께 실행하게 합니다.
    - 프로그램에 넘겨주는 인자는 하나로, 0번 인자에 path를 넣어줍니다.
    - 실행한 프로그램의 종료를 기다리지 않고 pmanager는 이어서 실행되어야 합니다.
    - 성공 시 별도의 메시지를 출력하지 않으며, 실패 시에만 메시지를 출력합니다.
    - 힌트: sh.c의 runcmd 함수가 BACK 명령에 대해 처리하는 방식을 참고하세요.
  - memlim <pid> <limit>
    - pid를 가진 프로세스의 메모리 제한을 limit으로 설정합니다.
    - 성공 여부를 출력합니다.
  - exit
    - pmanger를 종료합니다.

Process Manager

- 명령줄의 입력은 다음과 같은 형식을 항상 지킨다고 가정해도 좋습니다.
  - 모든 입력은 알파벳 대소문자, 숫자, 공백, 그리고 개행으로만 이루어집니다.
  - 명령의 맨 앞이나 맨 뒤에는 공백이 없습니다.
  - 명령과 옵션, 옵션과 옵션 사이에는 정확히 하나의 공백이 주어집니다.
  - pid, stacksize, limit은 0 이상 10억 이하의 정수입니다.
  - path는 길이가 50을 넘지 않으며, 알파벳 대소문자와 숫자로만 이루어진 문자열입니다.
    - 널 문자 등이 들어갈 공간을 확보하기 위해 배열의 크기는 넉넉하게 잡을 것을 권장합니다.
  - 각 명령은 해당 명령의 형식을 항상 따릅니다.
  - 명세에 주어지지 않은 명령은 입력되지 않습니다.

주의사항

- 스케줄러는 Project 1에서 구현한 스케줄러가 아닌, xv6에 기본으로 들어있는 round robin 스케줄러를 사용합니다.
  - Project 2 수행 중 스케줄러에서 수정되어야 할 것이 있다면 기본 스케줄러 코드를 수정하면 됩니다.
- fork 시스템 콜을 통해 만들어진 자식 프로세스는 그 시점에서 부모 프로세스가 가지고 있던 mode, memory limit 및 stack size 설정을 그대로 가지고 가야 합니다.
- exec 및 exec2 시스템 콜을 통해 만들어진 프로세스는 mode와 memory limit이 기본값으로 만들어져야 합니다. exec의 경우 stack size가 기본값인 1로 만들어져야 합니다.

팁

- Race condition이 발생하지 않도록 lock을 적절하게 사용하기를 권장합니다.
- pmanager 구현 시 sh.c의 구현을 참고하면 좋습니다.
- ulib.c에 있는 함수들을 사용하면 편리합니다.
- 제공해드린 테스트 파일들 이외에도 과제의 명세를 모두 따르는지 확인하기 위한 다양한 테스트 프로그램을 직접 제작해보실 것을 권장합니다.

평가지표

평가 기준은 다음과 같으며, 각 명세의 일부분을 완성하였다면 부분점수가 주어집니다.
실패하는 테스트가 있을 시 해당 부분에서 감점이 이루어집니다.

평가 기준	배점	비고
Administrator mode	10	Process manager 실행 중 다른 기능의 부실로 인해 발생한 문제는 해당 부분에서 감점이 이루어지며, process manager에 대한 평가는 pmanager 프로그램 자체의 동작에 대해서만 이루어집니다.
Custom stack size	15	
Per-process memory limit	15	
Shared memory	25	
Process manager	15	
Wiki	20	
총점	100	

평가

- **Completeness**: 명세의 요구조건에 맞게 xv6가 올바르게 동작해야 합니다.
- **Defensiveness**: 발생할 수 있는 예외 상황에 대처할 수 있어야 합니다.
- **Wiki&Comment**: 테스트 프로그램과 위키를 기준으로 채점이 진행되므로 위키는 최대한 상세히 작성되어야 합니다.
- **Deadline**: 데드라인을 반드시 지켜야 하며, 데드라인 이전 마지막으로 commit/push된 코드를 기준으로 채점됩니다.
- **Do NOT copy or share your code.**

Wiki

- Wiki에는 다음의 항목들이 서술되어야 합니다.
  - **디자인**: 각 스케줄러를 어떻게 구현해야 하는지, 추가/변경되는 컴포넌트들이 어떻게 상호작용하는지, 이를 위해 어떠한 자료구조들이 필요한지 등을 서술합니다.
  - **구현**: 실제 구현과정에서 변경하게 되는 코드영역이나 작성한 자료구조 등에 대하여 서술합니다.
  - **실행 결과**: 각 스케줄러가 올바르게 동작하고 있음을 확인할 수 있는 실행 결과와 이에 대한 설명을 서술합니다.
  - **트러블슈팅**: 코드 작성 중 생겼던 문제와 이에 대한 해결 과정을 서술합니다.
- 스케줄러에 대한 유의미한 그래프나 구조도 등 위키를 풍부하게 만들어주는 요소에는 추가 점수가 있을 수 있습니다.

테스트 프로그램

- 작성한 스케줄러가 올바르게 동작하는지 테스트하기 위한 테스트 프로그램이 업로드 되었습니다.
- 이는 과제의 편의성을 위한 것으로 제공되는 테스트 프로그램 이외의 테스트가 존재할 수 있습니다.
- 테스트는 명세를 넘어가는 기능을 요구하지 않으며, 몇몇 예외상황에 대한 테스트는 있을 수 있습니다.

제출

- 제출은 Hanyang gitlab을 통해 이루어져야 합니다.
- 제출된 repository는 반드시 그림의 디렉토리 구조를 가져야 합니다. (os_practice가 repo폴더)
- xv6-public폴더의 이름이 지켜지면 되며, 파일이나 폴더가 추가되어도 상관없습니다.
- 채점은 과제 마감 시간 전에 마지막으로 commit 및 push된 버전을 기준으로 합니다. 만일을 위해 commit 내역을 삭제하지 않도록 해주시기 바랍니다.