Project #2

Dept. of Computer Science

Hanyang University

Process: Process Management & LWP

Goal of project

- xv6의 프로세스는 일반적인 운영체제에서 필요한 최소한의 기능을 가지고 있습니다.

- Xv6에서 Threading과 Process Management를 제공하지 않습니다. 해당 기능들을 추가하여 xv6를 개선합니다.

- 기본 구조에 보다 다양한 기능들을 추가하고 프로세스들을 쉽게 관리할 수 있는 도구를 만들어 xv6를 더욱 풍부하게 만들어 봅시다.

Requirements

- **Process with various stack size**
  - int exec2(char *path, char **argv, int stacksize);
- **Process memory limitation**
  - int setmemorylimit(int pid, int limit);
- **Process manager**
  - List
  - Kill
  - Execute
  - Memlim
  - Exit
- **Pthread in xv6**
  - int thread_create(thread_t *thread, void *(*start_routine)(void *), void *arg);
  - int thread_join(thread_t thread, void **retval);
  - void thread_exit(void *retval);
  - Other system calls(fork, exec, wait…)

Process Management

Process with various stack size

- xv6는 exec 시 하나의 스택용 페이지와 하나의 가드용 페이지를 할당합니다.
- 스택용 페이지를 원하는 개수만큼 할당받는 exec2 시스템 콜을 만듭니다.

Process with various stack size

- int exec2(char *path, char **argv, int stacksize);
  - 스택용 페이지를 여러 개 할당받을 수 있게 하는 시스템 콜입니다.
  - 첫 번째와 두 번째 인자의 의미는 기존 exec 시스템 콜과 동일합니다.

- stacksize에는 스택용 페이지의 개수를 전달받습니다.
  - stacksize는 1 이상 100 이하의 정수여야 합니다.
  - 가드용 페이지는 이 크기에 관계없이 반드시 1개를 할당해야 하고, 가상 주소상 스택 페이지들의 바로 아래에 있어야 합니다.

Process memory limitation

- 특정 프로세스에 대해 할당받을 수 있는 메모리의 최대치를 제한합니다.
- 유저 프로세스는 추가적인 메모리를 할당받을 때 그 프로세스의 memory limit을 넘는 경우 할당받지 않아야 합니다.
- 프로세스가 처음 생성될 때에는 제한이 없으며, 이후 시스템 콜을 통해 이 제한을 설정할 수 있습니다.

Process memory limitation

- **int setmemorylimit(int pid, int limit);**
  - **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.**
  - **기존 할당받은 메모리보다 limit가 작은 경우 -1을 반환합니다.**
  - **pid가 존재하지 않는 경우, limit이 음수인 경우 등에도 -1을 반환합니다.**
  - **정상적으로 동작했다면 0을 반환합니다.**

Process manager

- 현재 실행 중인 프로세스들의 정보를 확인하고 관리할 수 있는 유저 프로그램을 만듭니다.
- 실행 후 종료 명령어가 들어올 때까지 한 줄씩 명령어를 입력받아 동작합니다.
- 프로그램의 이름은 pmanager로 합니다.

Process manager

- pmanager는 다음과 같은 명령어들을 지원해야 합니다.

- list
  - 현재 실행 중인 프로세스들의 정보를 출력합니다.
  - 각 프로세스의 이름, pid, 스택용 페이지의 개수, 할당받은 메모리의 크기, 메모리의 최대 제한이 있습니다.
  - Process가 실행한 Thread의 정보를 고려하여 출력하여야 합니다.
  - Thread의 경우 출력하지 않습니다.
  - 프로세스의 정보를 얻기 위한 시스템 콜은 자유롭게 정의해도 됩니다.

- kill
  - pid를 가진 프로세스를 kill합니다. kill 시스템 콜을 사용하면 됩니다.
  - 성공 여부를 출력합니다.

Process manager

- **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.**

- **memlim <pid> <limit>**
  - **pid를 가진 프로세스의 메모리 제한을 limit으로 설정합니다.**
  - **limit의 값은 0 이상의 정수이며, 양수인 경우 그 크기만큼의 limit을 가지이고 0인 경우 제한이 없습니다.**
  - **프로세스의 메모리는 thread의 메모리를 고려해야 합니다.**
  - **성공 여부를 출력합니다.**

- **exit**
  - **pmanager를 종료합니다.**

Process manager

- 명령줄의 입력은 다음과 같은 형식으로 들어와야 합니다.
  - 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 실행되지 않습니다.

LWP

Light-weight process

- 프로세스는 서로 독립적으로 실행되고, 자원을 공유하지 않으며 서로 별개의 주소 공간과 파일 디스크립터를 가집니다.
- Pthread는 유닉스 계열의 운영체제에서 병렬적인 프로그램을 작성하는 데에 주로 사용되는 API로, LWP의 대표적인 구현체 중 하나입니다.
- Light-wight process (LWP)는 다른 LWP와 자원과 주소 공간 등을 공유하여 유저 레벨에서는 멀티태스킹을 가능하게 해주는 개념입니다.
- xv6에서는 Process만 구현되어 있습니다. 이에 Thread기능을 하는 LWP를 구현하는 것을 목표로 합니다.

Specification - API (thread_create)

- int thread_create(thread_t *thread, void *(*start_routine)(void *), void *arg);
  - 새 스레드를 생성하고 시작합니다.
  - thread : 스레드의 id를 지정합니다.
  - start_routine : 스레드가 시작할 함수를 지정합니다.
    새로운 스레드는 start_routine이 가리키는 함수에서 시작하게 됩니다.
  - arg : 스레드의 start_routine에 전달할 인자입니다.
  - return : 스레드가 성공적으로 만들어졌으면 0, 에러가 있다면 -1을 반환합니다.

Specification - API (thread_exit)

- **void thread_exit(void *retval);**
  - **스레드를 종료하고 값을 반환합니다.**
  - **모든 스레드는 반드시 이 함수를 통해 종료하고,**
    시작 함수의 끝에 도달하여 종료하는 경우는 고려하지 않습니다.
  - **retval : 스레드를 종료한 후 join 함수에서 받아갈 값입니다.**

Specification - API (thread_join)

- int thread_join(thread_t thread, void **retval);
  - 해당 스레드의 종료를 기다리고, 스레드가 thread_exit을 통해 반환한 값을 반환합니다.
  - 스레드가 이미 종료되었다면 즉시 반환합니다.
  - 스레드가 종료된 후, 스레드에 할당된 자원들을 회수하고 정리해야 합니다.
    (페이지 테이블, 메모리, 스택 등)
  - thread : join할 스레드의 id입니다.
  - retval : 스레드가 반환한 값을 저장해 줍니다.
  - return : 정상적으로 join했다면 0을, 그렇지 않다면 -1을 반환합니다.

Specification – System Call

- 스레드가 xv6에서 하나의 프로세스로서 올바르게 동작하려면
  여러 시스템 콜에서 스레드를 지원하여야 합니다.

- 다음과 같은 시스템 콜들에 대해 스레드가 잘 동작하는지를 평가합니다.
  동시에 여러 스레드에서 시스템 콜을 호출하더라도 정상적으로 동작해야 합니다.

Specification – System Call

- **fork**
  - 스레드에서 fork가 호출되면 기존의 fork 루틴을 문제 없이 실행해야 합니다.
  - 해당 스레드의 주소 공간의 내용을 복사하고 새로운 프로세스를 시작할 수 있어야 합니다.
  - wait 시스템 콜로 기다릴 수도 있어야 합니다.

- **exec**
  - exec가 실행되면 기존 프로세스의 모든 스레드들이 정리되어야 합니다.
  - 하나의 스레드에서 새로운 프로세스가 시작하고 나머지 스레드는 종료되어야 합니다.

- **sbrk : sbrk는 프로세스에게 메모리를 할당하는 시스템 콜입니다.**
  - 여러 스레드가 동시에 메모리 할당을 요청하더라도 할당해주는 공간이 서로 겹치면 안 되며 올바르게 할당할 수 있어야 합니다.
  - sbrk에 의해 할당된 메모리는 프로세스 내의 모든 스레드가 공유 가능합니다.

Specification – System Call

- **kill : kill은 Process를 종료시키는 시스템 콜입니다.**
  - **하나 이상의 스레드가 kill 되면 프로세스 내의 모든 스레드가 종료되어야 합니다.**
  - **kill 및 종료된 스레드의 자원들은 모두 정리되고 회수되어야 합니다.**

- **sleep : sleep은 특정 시간만큼 process를 sleep시키는 시스템 콜입니다.**
  - **한 스레드가 sleep을 호출하면 그 스레드만 sleep 해야 합니다.**
  - **자고 있는 상태에서도 kill에 의해서 종료될 수 있어야 합니다.**

- **pipe**
  - **각 스레드에서 각각 화면에 출력하는 데에 문제가 없어야 합니다.**

Tips & Evaluation

Tips

- xv6에 기본 Scheduler를 사용합니다.
  - Project 2 수행 중 스케줄러에서 수정이 필요하다면, 기본 코드를 수정하면 됩니다.
  - 다른 프로세스들과 마찬가지로 스레드들도 스케줄러에 의해 RR로 스케쥴링됩니다.

- 어떤 부분을 어떻게 고쳐야 할지 막막하다면, xv6 코드를 참조하시기 바랍니다.
  - fork, exec, exit, wait 등의 구현내용을 분석해보시면 과제 진행에 도움이 됩니다.

- git clone을 새로 받은 후, 과제를 진행하셔도 무방합니다.

- GitLab에 새로운 디렉토리를 생성하셔도 괜찮습니다. (ex. project02)
  - 기존 Project1내용을 별도 디렉토리에 저장하셔도 괜찮습니다. (ex. project01)

Tips

- Process Management는, Process와 연관이 높습니다.
  - proc.c proc.h 등 관련 파일을 참고하시기 바랍니다.
  - Execute명령에서, sh.c의 runcmd 함수가 BACK 명령에 대해 처리하는 방식을 참고하세요.

- fork 시스템 콜을 통해 만들어진 자식 프로세스는 그 시점에서
  부모 프로세스가 가지고 있던 memory limit, stack size를 그대로 복사합니다.

- Race condition이 발생하지 않도록 lock을 적절하게 사용하기를 권장합니다.
  - 스레드 관련 작업 중에는 ptable에 대한 lock을 고려하는 것을 권장드립니다.

Tips

- LWP는 xv6에서 프로세스와 비슷하게 취급됩니다.
  즉, 과제에 명시된 일부 기능을 제외하고는 기본적으로 프로세스처럼 동작합니다.
  - 스레드는 프로세스와 유사합니다.
  - 단, 스레드는 모두 main thread의 pid를 공유합니다.

- 프로세스의 페이지 테이블을 공유하는 것으로, 스레드 간에 주소 공간을 공유할 수 있습니다.

- 정상적인 스레드의 구현 및 동작은, 리눅스에서 pthread의 구현과 동작을 참고하시기 바랍니다.

Evaluation

- **Code**
  - 명세의 요구 조건을 모두 올바르게 구현해야 합니다.
  - 코드에는 주석이 함께 작성되어야 합니다.

- **Wiki**
  - 테스트 프로그램과 위키를 기준으로 채점됩니다.
  - 위키는 실제로 동작하는 장면과 함께 본인의 디자인에 대해 상세히 작성되어야 합니다.

- **Submission**
  - 데드라인을 반드시 지켜야 합니다.
  - 데드라인 이전에 GitLab에 마지막으로 commit/push 된 코드를 기준으로 채점됩니다.

Code

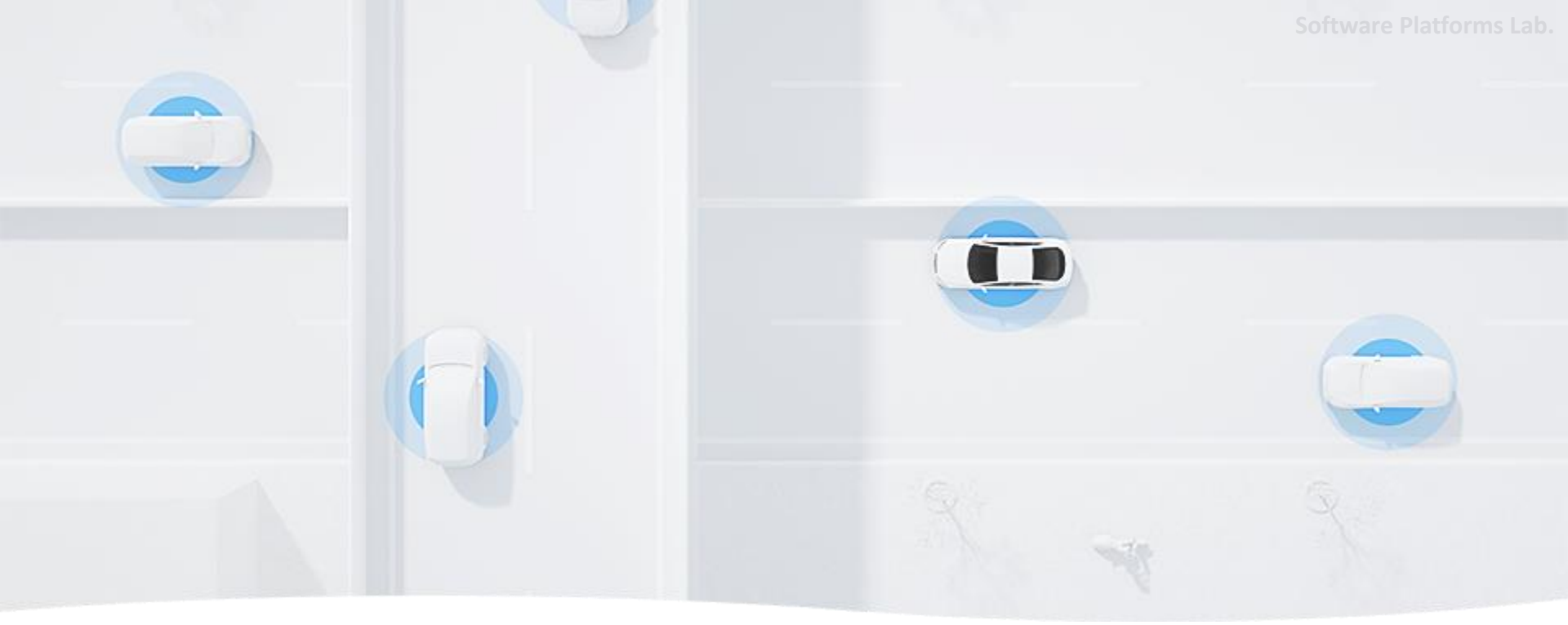
- **Completeness**
  - 명세의 요구 조건을 모두 올바르게 구현해야 합니다.
- **Defensiveness**
  - 발생할 수 있는 예외 상황에 대처할 수 있어야 합니다.
- **Comment**
  - 코드에는 반드시 주석이 있어야 합니다.
- **DO NOT ASK OR SHARE CODE !!**

Wiki

- **Design**
  - 명세에서 요구하는 조건에 대해서 **어떻게 구현할 계획**인지, 어떤 자료구조와 알고리즘이 필요한지, **자신만의 디자인**을 서술합니다.
- **Implement**
  - 본인이 **새롭게 구현하거나 수정한 부분**에 대해서 무엇이 기존과 어떻게 다른지, 해당 코드가 무엇을 목적으로 하는지에 대한 설명을 구체적으로 서술합니다.
- **Result**
  - 컴파일 및 **실행 과정**과, 해당 명세에서 요구한 부분이 정상적으로 동작하는 **실행 결과**를 첨부하고, 이에 대한 동작 과정에 대해 설명합니다.
- **Trouble shooting**
  - 과제를 수행하면서 **마주하였던 문제**와 이에 대한 **해결 과정**을 서술합니다. 혹여 문제를 해결하지 못하였다면 어떤 문제였고 어떻게 해결하려 하였는지에 대해서 서술합니다 .
- **And whatever you want ☺**

Submission

- You should upload your code to HYU GitLab repository.
- You should upload your wiki to HYU LMS (only .pdf)
- Wiki file name is “ELE3021_ project02_[classNO]_ [studentID].pdf”
- **Due date : 2023. 05. 28. 23:59 (No late submission allowed)**



Thank you :)