1. 소개

위의 과제는 linux 환경에서 프로세스 실행 / 종료를 자동을 관리하는 프로그램에서 추가적인 기능을 구현하는 것을 목표로 하고 있습니다.
기존에 sigaction 함수를 이용하여 SIGCHLD 시그널을 처리하였는데 이를 SIGNALFD()를 이용하여 처리하였고, 프로그램 실행 정보에 ORDER항목을 추가하는데 성공하였습니다.
구현 및 실행 / 테스트는 노트북에 설치되어 있는 linux 운영체제를 사용하였습니다.

2. 관련 연구

2.1 소스 분석

Action type : ONCE와 RESPAWN으로만 구성된 열거형 데이터 타입

Task type : 프로세스에 대한 정보를 갖는 구조체

strstrip : :를 기준으로 해서 한 줄에서 원하는 부분만을 추출하기 위한 함수이다.

check_valid_id : 읽어들인 id가 유효한 id인지 확인하기 위한 함수이다.

lookup_task : 읽어들인 id의 task가 tasks에 있는지 확인하고 존재한다면 tasks에 존재하는 task를 반환해주고 없으면 NULL값을 반환하는 함수이다

append_task : 새로운 Task를 위한 공간을 할당하고 Linked list로 연결되어 있는 Tasks에서 적절한 위치에 Task를 연결하는 역할을 하는 함수

read_config : file을 읽어들여 비어있거나 #에 의해 주석처리 되어있는 줄의 체크와 : 을 기준으로 id, action, order, pipe-id, command를 Task의 적절한 Member에 넣어주기 위한 함수

make_command_argv : 문자열의 유무를 확인하고 문자열이 있다면 명령어의 복사본을 메모리를 할당하여 만들어주는 함수이다

spawn_task(Task) : pipe함수를 이용하여 파이프를 생성하고 fork() 함수를 통해 system call을 하여 자식 프로세스를 생성하고 execvp 함수를 호출하여 자식 프로세스에서 command를 실행한다.

spawn_task(void) : Tasks에서 running상태이면서 task가 비어있지 않은 위치까지 이동하여 함수 내에서 선언한 task를 이용한 spawn_task(Task)를 호출한다.

wait_for_children : waitpid(-1, NULL, WNOHANG)를 호출하여 임의의 자식 프로세스를 기다리며 멈추거나 상태가 보고되지 않은 자식들을 위해 리턴을 해준다. 또한 Task의 action이 ACTION_RESPAWN일 경우 spawn_task를 호출한다.

terminate_children : signo를 받아들여 kill함수를 통해 자식 프로세스를 종료시키는 함수이다.

2.2 System call & API

signalfd() : 호출자를 대상으로하는 signal를 받아들이는 데 사용할 수있는 file descriptor을 만든다.

fork() : 새로운 프로세스를 만드는 함수인데 현재 실행되는 프로세스에 대해 복사본 프로세스를 생성하는 방식을 사용한다. 이때 원본 프로세스를 parent process라고 부르고 복제된 프로세스를 child process라고 부른다.

Kill() : 프로세스에 특정 시그널 번호를 전송하는 함수

memset() : 첫번째 인자로 시작하는 메모리 주소 부터 세번째 인자 개수의 바이트를 두번째 인자 값으로 채운다.

pipe() : 프로세스 사이의 통신에서 사용하는 파이프를 생성한다. 생성되는 파이프는 프로세스 안에 생성되는 것이 아니라 커널에 생성되며 프로세스에는 파이프를 이용할 수 있는 file descriptor만 제공한다.

execvp() : 첫번째 인자에 지정한 파일을 실행하며 두번째를 인자로 전달한다.

waitpid() : 인수로 주어진 pid 번호의 자식프로세스가 종료되거나, 시그널 함수를 호출하는 신호가 전달될때 까지 waitpid 호출한 영역에서 일시 중지 된다.

exit() : 현재의 C프로그램 자체를 완전 종료한다.

memmove() : 두번째 인수가 가리키는 곳 부터 세번째 인수 바이트 만큼을 첫 번째 인수가 가리키는 곳으로 옮긴다. [메모리 영역 복사]

malloc() : 메모리 할당을 위한 system call

3. 추가 기능 구현 방법

3.1 sigaction() 대신 signalfd() 함수를 이용하여 SIGCHLD 신호 처리

(1) sigaddset(&sa.sa_mask, SIGCHLD)를 이용하여 sa.sa_mask에 SIGCHLD 신호 추가해 준다.

(2) s_fd = signalfd(-1, &sa.sa_mask, 0) 를 통해 sa.sa_mask에 있는 SIGCHLD를 받아 들이는 데 사용할 수 있는 file descriptor을 새로 만들어 s_fd에 삽입한다.

(3) main 내부의 while(!terminated)내에 read()함수를 통해 읽어들인다

(4) if(fdsi.ssi_signo == SIGCHLD)에서 fds.ssi_signo가 SIGCHLD이면 wait_for_children함수를 호출하여 SIGCHLD를 handeling한다.

(5) SIGCHLD가 잘 처리되었는지 확인을 위해 출력문을 추가해준다.

3.2 order를 추가하여 순서지정을 가능하게 한다.

(1) config1.txt에 action pipe_id 사이에 order를 위한 field를 추가한다.

(2) Task 구조체에 order attribute를 추가한다.

(3) read_config함수에 order를 처리하기 위한 부분을 추가한다

(3).1 공백일 경우 rand()%10000를 통해 4자리 랜덤 순자를 배정하여 임의의 순서대로 실행되도록 한다

(3).2 4자리가 넘는 숫자가 있다면 예외처리 해준다.

(4) append_task함수 내에서 Linked list를 생성할 때 order순서에 맞게 연결되면서 추가되도록 코드를 작성한다.