4장 과제

1731017 김민경

1. 운영체제가 IPC (Interprocess Communication)를 지원하게 된 이유는 무엇인가?

프로세스들의 주소 공간은 완전히 분리되어 있어서, 두 프로세스 사이에는 코드를 제외한 어떤 메모리 공간도 공유되지 않는다. 프로세스가 다른 프로세스의 메모리에 접근할 수 없기 때문에, 응용 프로그램 수준에서 두 프로세스 사이의 통신은 불가능하다. 운영체제 커널은 프로세스들이 서로 통신할 수 있는 방법을 제공하는데, 이를 IPC 라고 한다.

2. IPC의 방법 없이 프로세스들이 서로 데이터 혹은 정보를 주고 받을 수 있는가? 있다면 어떻게 없다면 왜?

IPC 방법 없이 프로세스들이 서로 데이터나 정보를 주고받을 수 있다. IPC 방법은 코딩이나 디버깅이 어렵고 통신에 따른 여러 오버헤드가 존재한다. 따라서 최근에는 거의 사용하지 않으며, 코딩하기 쉽고 통신에 따른 오버헤드가 적은 멀티스레딩 방법을 사용한다.

3. 신호 핸들러는 왜 반드시 사용자 모드에서 실행되는가?

신호 핸들러는 사용자가 작성한 코드이다. 이 때, 신호 핸들러가 커널 모드에서 실행될 경우 커널 영역의 코드를 바로 접근할 수 있게 된다. 만약 신뢰성이 떨어지는 코드를 작성했을 경우 커널 코드가 바로 실행되어서 시스템 전체가 훼손될 수 있는 가능성이 있다. 따라서 신호 핸들러는 사용자 모드에서 실행되어야 한다.

4. 신호 핸들러를 등록하는 시스템 호출과 신호를 보내는 시스템 호출은 무엇인가?

신호 핸들러를 등록하는 시스템 호출로는 signal(int sig, void (*handler)(int)) 가 있다. 이는 handler() 함수를 sig 번호의 신호 핸들러로 등록하는 것이다.

신호를 보내는 시스템 호출로는 kill과 raise 가 있다. 먼저, kill(pid_t pid, int sig)는 pid 번호의 프로세스에게 sig 신호를 보내는것이다. 다음으로 raise(int sig)는 자기자신에게 sig 신호를 보내는 함수로, 싱글 스레드 프로그램의 경우 kill(getpid(), sig)와 동일하며, 멀티스레드 프로그램의 경우 pthread_kill(pthread_self(), sig)와 동일하다.

5. 신호가 도착해 있는 경우 신호 핸들러가 실행되는 시점을 설명하라.

첫째, 신호를 수신한 프로세스가 현재 스케줄링을 기다리는 준비상태에 있는경우이다. 프로세스는 커널에 의해 스케줄링되어 실행되기 바로 직전 사용자 모드에서 신호 핸들러를 실행한다.

둘째, 프로세스가 어떤 것이든 상관없이 시스템 호출의 실행을 마친후 커널모드에서 사용자 모드로 바뀌고 프로세스의 사용자 코드로 돌아가기 직전에 신호 핸들러를 실행한다.

6. 신호를 보내는 즉시 수신 프로세스가 신호를 받는가? 그렇다면 그 이유를 설명하고, 그렇지 않다면 또 그 이유를 설명하라.

신호가 발생한 즉시 프로세스의 신호 핸들러가 실행되는 것은 아니다. 신호 발생과 신호 핸들러 실행 사이에는 시간 차가 존재한다.
프로세스에서 아무 시스템 호출이라도 실행된 후 사용자 모드로 돌아갈 때, 혹은 커널 스케줄러가 실행할 프로세스를 선택하여 사용자 모드로 바꾼 후, 사용자 코드가 실행되기 전에 신호 핸들러가 실행된다. 또, 신호 핸들러는 커널에서 수신 프로세스를 스케줄 할 때만 실행된다. 그러므로 프로세스가 현재 입출력 대기 상태로 종단된 경우, 입출력이 완료될 때까지 수신된 신호는 처리되지 못한 상태로 있게된다.

7. 부모 자식 프로세스 사이에서 데이터를 주고받기 위해 가장 유용한 IPC 방법은? 공유메모리, 신호, 익명의 파이프, FIFO 중에서 어떤 것이며, 그 이유는 무엇인가?

부모 자식 프로세스 관계에서 가장 유용한 IPC 방법은 익명 파이프이다.
부모가 자식을 FORK()로 생성하는 경우 스택, 환경 변수 등을 상속받게 된다. 그중에서 파일 디스크립터도 존재한다. 파일 디스크립터에는 익명 파이프의 정보도 포함되어 있어서 부모와 자식은 익명 파이프를 공유한다. FIFO는 부모 자식 간에도 이용이 가능하나 별도의 파이프 생성 구문을 추가해주어야 하며, 익명 파이프와 다르게 커널 내 FIFO 파일 공간이 추가적으로 할당되기 때문에 부모 자식 간 데이터를 주고 받는 경우에는 익명 파이프에 비해 효율적이지 못하다.
공유메모리는 사용자 영역에 별도의 공간을 추가적으로 할당해야하며 공유메모리를 구현하는 코드는 파이프에 비해 복잡하다. 신호는 별도의 신호 처리를 위한 절차가 필요하고 정의상에 많은 표현을 가질할 수 없어 부모 자식 간 데이터가 다양한 경우 효율적이지 못하다.

8. 익명의 파이프와 FIFO의 차이점을 2가지만 들어보라.

익명의 파이프는 부모 프로세스와 자식 프로세스 간 파일 디스크립터를 통해 통신하므로 파이프 이름이 없는 반면에 FIFO는 서로 무관한 프로세스들 사이의 통신을 하게 되므로 파일 이름을 가지고 있다. 또, 익명의 파이프는 파이프의 양 끝단에서 부모 자식 간 파일 디스크립터를 상속받았기에 읽고 쓰는 것이 동시에 가능하나 이름을 가진 파이프는 양끝단에서 읽고 쓰기가 동시에 불가능하다. 따라서 읽기 | 쓰기 모드 둘 중 하나로 선택해서 사용해야 한다.

9. 멀티스레딩을 이용하여 다중작업을 개발할 수 있다면, IPC를 이용하여 프로세스 사이에 통신하는 방법은 필요없는것일까? 필요있다고 생각하면 사례를 들어 설명하고, 필요 없다고 생각하는 그 이유를 설명하라.

IPC는 프로세스와 프로세스간 통신방법을 정의한 개념이지만 스레드와 스레드간 통신이 필요한 경우에도 필요하다. 물론 멀티 스레드 환경에서 스레드들 간 통신은 프로세스 생성, 컨텍스트 스위칭에 의해 생성되는 많은 오버헤드가 발생되는 것을 줄여주기 때문에 IPC의 문제점을 해결해줄 수 있다.
하지만 A 프로세스의 스레드가 B. 프로세스의 스레드를 참고해야하는 경우에는 IPC가 필요하다. 예를 들어 웹에서 회원가입을 해야한다고 가정해보자.
이와 같은 경우에 web container 내 servlet 과 database 간 데이터 송수신이 필요하다. 구체적으로 소켓이나 파이프을 이용해 포트번호 등을 맞추고 연결하는 등의 IPC 통신 과정이 필요하다. 이처럼 스레드가 특정 스레드의 스택 공간에 대한 정보가 필요한 경우 IPC를 구현하여 정보 공유를 할 필요성이 있다.

10. 공유메모리, 신호, 파이프를 사용하는 IPC 프로그래밍을 위해 필요한 시스템 호출 함수를 들고 각각 한 줄씩 간단히 설명하라.

공유메모리에는 다음과 같은 시스템 호출 함수들이 있다.
shm_open()은 공유메모리 객체를 생성하거나 이미 존재하는 공유메모리 객체를 연다. ftruncate()는 공유메모리의 크기를 지정한다. mmap()은 공유메모리의 0번지 부터 SIZE 바이트 만큼을 현재 프로세스의 주소 공간에 매핑하여, 공유 메모리를 엑세스할 수 있는 주소를 리턴한다. munmap()은 공유메모리 매핑을 해제한다. close()는 공유메모리를 닫아 연결을 끊는다. 마지막으로 shm_unlink()는 공유메모리의 이름을 제거하여 완전히 없앤다. 하지만, 현재 공유 메모리를 사용하고 있는 프로세스가 있다면 (참조 카운터가 1 이상) 모두 종료 시 까지 공유메모리를 없애지 않고 둔다.

신호는 다음과 같은 시스템 호출 함수들이 있다.
signal (int sig, void (*handler)(int)) 는 handler() 함수를 sig 번호의 신호의 핸들러로 등록한다. raise (int sig)는 프로세스 자신이나 스레드 자신에게 sig 신호를 보낸다. 등록된 핸들러가 있다면, 핸들러를, 없다면 디폴트값을 실행시킨다. kill (pid_t pid, int sig)는 pid 번호의 프로세스에게 sig 신호를 보낸다. pause() 는 프로세스가 신호를 수신할 때까지 대기한다.

마지막으로 파이프는 다음과 같은 시스템호출함수들이 있다.
pipe() 는 프로세스 사이 데이터 통신을 위한 단방향 익명 파이프를 생성한다.
int mkfifo (const char* pathname, mode_t mode) 는 pathname 이름으로 mode 의 모드로 작동하는 이름을 가진 양방향 FIFO를 생성한다. open()은 FIFO() 의 해당 파일명과 특정모드를 지정해서 연다. read() / write() 는 파이프를 읽거나 쓴다. 그리고 close() 는 프로세스가 파이프에 접근하는 파일 디스크립터를 닫고자 할때 사용한다.