Process

**1. 프로세스의 개념**

* 프로세스는 cpu와 메모리를 할당 받아 **진행중인 프로그램이다**.
* 프로세스들마다 고유의 프로세스 ID(PID)를 할당 받는다.
  + 가장 먼저 실행되는 프로세스는 init이다. Init의 PID는 1이다.
  + 프로세스 init에 의해 다른 모든 프로세스들이 시작된다.

**2. 프로세스의 유형**

* 포그라운드 프로세스와 백그라운드 프로세스
  + 프로세스는 실행 형태에 따라 포그라운드(foreground) 프로세스와 백그라운드(background) 프로세스로 나뉜다

|  |  |
| --- | --- |
| **포그라운드 프로세스**  **(Foreground)** | 사용자와 상호 작용하는 프로세스  터미널에 직접 연결되어 입출력을 주고받는 프로세스  명령 입력 후 수행 종료까지 기다려야하는 프로세스  화면에서 실행되는 것이 보이는 프로세스  Ex) 응용 프로그램, 명령어 등 |
| **백그라운드 프로세스**  **(Background)** | 사용자와 직접적인 대화를 하지 않고 뒤에서 실행되는 프로세스  사용자의 입력과 관계없이 실행되는 프로세스  실행은 되지만 화면에 나타나지 않고 실행되는 프로세스 |

* fork()와 exec()
  + 사용자가 새로 **프로세스를 생성하기 위해 사용하는 시스템 호출 함수**로는 fork()와 exec()가 있다.
  + fork() 함수
    - 새로운 프로세스를 만들 때 기존의 프로세스를 복제하는 방식을 사용하한다.
    - 새로운 프로세스를 위한 메모리를 할당한다.
    - 새로 생성된 프로세스는 원래의 프로세스와 똑 같은 코드를 갖고 있다.
    - 원본 프로세스를 부모 프로세스(parent process)라고 부르고, 새로 복제된 프로세스를 자식 프로세스(child process)라고 부른다.
  + exec() 함수
    - 호출하는 프로세스가 새로운 프로세스로 변경되는 방식이다.
    - 새로운 프로세스를 위한 메모리를 할당하지 않는다.
    - 호출한 프로세스의 메모리에 새로운 프로세스의 코드를 덮어 씌워버린다.
    - exec()를 호출한 프로세스가 아닌 exec()에 의해 호출된 프로세스만 메모리에 남게 된다.
* 데몬(Daemon)
  + 리눅스 시스템이 부팅 시 자동으로 실행되는 **백그라운드 프로세스**이다.
  + **메모리에 상주**하면서 사용자의 특정 요청이 오면 즉시 실행되는 대기 중인 **서버 프로세스**이다.
  + 주기적이고 지속적인 서비스 요청을 처리하기위해 실행된다.
  + 사용자들은 이 프로세스를 볼 수 있는 권한이 없다.
  + 리눅스에서 데몬을 실행하는 방법은 세가지가 있다.

|  |  |
| --- | --- |
| **Standalone**  **데몬** | 서비스가 요청이 들어오기 전에 메모리에 상주하는 단독 실행 방식  독립적으로 수행되며 서비스 요청에 응답하기 위해 항상 메모리에 상주  빠른 응답 속도를 요하는 경우에 사용  메모리에 항상 상주하므로 메모리 점유로 인한 서버 부하가 작음  실행 스크립트 위치는 “/etc/inetd.d/”  관련 서비스 : http, mysql, nameserver, sendmail |
| **inetd 데몬**  **(슈퍼 데몬)** | inetd는 다른 데몬들의 상위에 존재하는 standalone 데몬  ientd는 자체적으로 하나의 독립 데몬이지만 여러 가지 다른 서비스들을 제어하고 관리  보안상의 이유로 리눅스 커널 2.4버전부터 xinetd(extended inetd)가 역할을 대신 |
| **inetd 타입**  **데몬** | inetd 타입 데몬들은 직접 서비스를 가동하지 못하고 inetd 데몬이 활성화 되어야만 해당 서비스를 제공  Telnet, FTP, rloging과 같은 데몬들이 inetd 타입 데몬에 해당  Inetd 서비스 요청이 종료되면 inetd 타입 데몬들도 자동으로 종료 |

* 시그널(Signal)
  + 시그널은 프로세스에게 이벤트 발생을 전달해주는 소프트웨어 인터럽트이다.
  + 시그널이 프로세스에게 전달될 때 4가지 상황이 발생한다.(기본적인 시그널 처리방식)

|  |  |
| --- | --- |
| **SIG\_IGN (SIG\_PF)1** | 운영체제에 의해 시그널이 무시된다. |
| **SIG\_ERR (SIG\_PF)-1** | 운영체제는 프로그램을 강제로 종료한다. |
| **SIG\_DFL (SIG\_PF)0** | 지정한 시그널 처리 루팅을 실시한다. |
| **SIG\_HOLD (SIG\_PF)2** | 시그널이 블러킹된다. |

* + 명령어 kill -l로 시그널 리스트를 확인할 수 있다.
  + 지정된 이벤트의 시그널이 프로세스에 전달될 때 어떤 상황(기본 동작)이 발생하는지

|  |  |  |  |
| --- | --- | --- | --- |
| **번호** | **시그널** | **발생조건** | **기본 동작** |
| **1** | **SIGHUP** | 터미널과 연결이 끊어졌을 때 | 종료 후 재시작 |
| **2** | **SIGINT** | Crlt + C 입력 시 | 종료 |
| **3** | **SIGQUIT** | Crlt + ＼입력 시 | 코어 덤프 |
| **6** | **SIGABRT** | abort(비정상 종료) 함수에 의해 발생 | 코어 덤프 |
| **9** | **SIGKILL** | 프로세스 강제 종료 시 | 종료 |
| **13** | **SIGPIPE** | 종료된 소켓이 쓰기를 시도할 때 | 코어 덤프 |
| **14** | **SIGALRM** | 알람 타이머 만료 시 | 코어 덤프 |
| **15** | **SIGTERM** | Kill 시스템 호출 시 | 종료 |
| **17** | **SIGCHLD** | 자식 프로세스가 종료 시 | 무시 |
| **18** | **SIGCONT** | 중지된 프로세스 실행 시 | 무시 |
| **19** | **SIGSTOP** | SIGCONT 시그널을 받을 때까지 | 종료 |
| **20** | **SIGTSTP** | Crlt + Z 입력 시 | 프로세스 대기로 전환 |