**강의명: 유닉스 시스템**

**실습 번호: 8**

**실습 제목: 신호(Signals)**

**학생 이름: 황귀훈**

**학번: 201710885**

**1. 신호의 개념**

**1.1**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**1.2**

Signal은 어떤 event가 발생하였음을 알리기 위하여 process에게 전달되는 소프트웨어 인터럽트이다. Signal을 발생시키는 event의 종류에는 여러가지가 있는데 이번 실습에서는 단말기에서 발생하는 사용자 입력(^C)를 사용한다. Event에 의하여 signal이 “generate”되면 곧 process에게 “deliver”된다. 이 때 process에게 signal이 “deliver”되면 default action이 수행되거나 catch되거나 ignore될 수 있다.

1번 신호의 개념에 주어진 프로그램 loop.c는 for문을 통해 20번 반복하며 매 회 printf()함수를 사용하여 "sleeping for 1 second...\n"를 출력한 뒤 sleep(1)함수를 사용하여 1초간 프로세스를 정지하는 프로그램이다. 명령 줄 “./loop” 수행한 후, 이 프로그램이 종료하기 전에, 키보드에서 ^C를 입력하면 SIGINT signal이 “generate”되고 곧 process에게 “deliver”된다. SIGINT signal을 전달받은 process는 default action을 수행하는데, SIGINT의 default action은 “terminate”로 process를 죽이는 것이다. 따라서 명령 줄 “./loop” 수행한 후, 이 프로그램이 종료하기 전에, 키보드에서 ^C를 입력하면 1.1의 결과화면과 같이 즉시 프로세스가 종료된다.

**2. 신호의 처리**

**2.1**

**테이블이(가) 표시된 사진

자동 생성된 설명**

**2.2**

**#include <unistd.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <signal.h>**

**int main(void)**

**{**

**int i;**

**if (signal(SIGINT, SIG\_IGN)==SIG\_ERR) {**

**perror("signal");**

**exit(1);**

**}**

**for( i=0 ; i<20 ; i++) {**

**printf("sleeping for 1 second...\n");**

**sleep(1);**

**}**

**return 0;**

**}**

2번 신호의 처리는 앞 실습에서 작성한 프로그램 loop.c를 수정하여 이 프로그램이 수행되는 도중 키보드에서 ‘^C’를 입력하더라도 이를 무시하고 계속 수행되는 프로그램이다.

먼저 여러 헤더파일들을 include해야 하는데, sleep() 함수를 사용하기 위해 <unistd.h>를, exit() 함수를 사용하기 위해 <stdlib.h>를, printf()함수를 사용하기 위해 <stdio.h>를, 그리고, signal() 함수를 사용하기 위해 <signal.h>를 include 했다.

앞 실습의 loop.c는 정수 변수 i를 선언하고 for문을 사용하여 20번 반복하는 동안 매회 printf()함수를 사용하여 **"sleeping for 1 second...\n"**를 출력하고 sleep(1)함수를 사용하여 1초간 프로세스를 정지하는 프로그램이었다. 이번 실습에서는 이 프로그램이 수행되는 도중 키보드에서 ‘^C’를 입력하더라도 이를 무시하고 계속 수행되도록 수정하여야 한다.

프로그램 수행 도중 ‘^C’가 입력되면 SIGINT signal이 “generate”되고 곧 process에게 “deliver”되는데 이 때 signal event를 무시하도록 for문 앞에 signal()함수에 SIGINT와 SIG\_IGN을 인수로 주어 호출하였다. Signal()함수에 첫 번째 인수로 주어진 SIGINT는 두 번째 인수의 지시에 따라 수행하게 되는데 두 번째 인수인 SIG\_IGN의 의미는 주어진 signal을 ignore하는 것이다.

또한, signal()함수는 오류 시 SIG\_ERR를 return하기 때문에, if문을 사용하여 만약 오류가 발생하면perror()함수를 사용해 오류 메세지를 출력한 후, exit(1)함수를 사용해 프로그램을 오류로 종료했다.

다음과 같이 앞 실습의 loop.c를 수정한 뒤 명령 줄 “./ignore”를 수행한 후, 이 프로그램이 종료하기 전에, 키보드에서 '^C'를 여러 번 입력해도 프로그램이 종료되지 않는 것을 확인했다.

**3. 함수 alarm**

**3.1**

**#include <unistd.h>**

**#include <stdio.h>**

**#include <signal.h>**

**#include <stdlib.h>**

**void sigusr\_handler() {**

**exit(0);**

**}**

**int main(int argc, char \*argv[])**

**{**

**alarm((unsigned int)atoi(argv[1]));**

**if (signal(SIGALRM, sigusr\_handler)==SIG\_ERR) {**

**perror("signal");**

**exit(1);**

**}**

**for(;;) {**

**printf("running...\n");**

**sleep(1);**

**}**

**return 0;**

**}**

3번 함수 alarm은 표준 출력으로 매초 "running...\n"을 출력하는 프로그램으로 이 프로그램은 명령 줄 인수 argv[1]에 주어진 시간(second) 동안만 수행한 후 종료하여야 한다. 이 프로그램 작성을 위하여 시스템 호출 signal, alarm, sleep을 사용하여야 한다.

먼저 여러 헤더파일들을 include해야 하는데, sleep(), alarm()함수를 사용하기 위해 <unistd.h>를, exit() 함수를 사용하기 위해 <stdlib.h>를, printf()함수를 사용하기 위해 <stdio.h>를, 그리고, signal() 함수를 사용하기 위해 <signal.h>를 include 했다.

그리고 이 프로그램은 기본적으로 매초 "running...\n"을 출력해야 되기 때문에 for(;;) 무한루프 안에 printf()함수를 사용하여 "running...\n"를 출력하고, sleep(1)함수를 사용하여 1초 프로세스를 정지하도록 하였다.

다음으로 명령 줄 인수 argv[1]에 주어진 시간(second) 동안만 수행한 후 종료하기 위해서는 주어진 시간 뒤에 signal이 발생하여 프로세스에 전달되면 프로세스를 종료하도록 해야 한다.

그러기 위해 먼저 alarm()함수를 사용했다. 함수 alarm()은 주어진 시간(초)가 지난 후 SIGALRM이 자신의 process에게 전달되게 한다. Alarm()함수의 첫 번째 인수는 unsigned int형 이기 때문에 명령 줄 인수 argv[1]에 주어진 string값을 atoi()함수를 사용하여 정수형으로 변환한 뒤 unsigned int형으로 변환했다.

alarm()함수에 의해 SIGALRM signal이 process에 전달되면 프로세스가 종료할 수 있도록 signal()함수에 첫 번째 인수로 SIGALRM를 주고 두 번째 인수로 sigusr\_handler를 주어 호출했다. 이 때 Signal()함수는 SIGALRM signal을 받으면 sigusr\_handler()함수를 수행하는데 sigusr\_handler()함수는 exit(0)을 수행하여 프로그램을 종료하도록 했다. 이 때 만약 signal()함수의 return값이 SIG\_ERR라면 perror()함수를 사용하여 오류 메시지를 출력하고 exit(1)함수를 사용하여 프로그램을 오류로 종료했다. 따라서 명령 줄 인수 argv[1]에 주어진 시간(second)뒤 alarm()함수에 의해 SIGALRM signal이 process에게 전달되면 signal()함수에 의해 sigusr\_handler()가 수행되고 이 함수는 exit(0)을 수행하므로 프로그램이 종료하게 된다.

그 결과 명령 줄 "date; ./alarm 5; date”를 수행하면 date명령을 수행하고 5초동안 "running...\n"을 출력한 뒤 다시 date명령을 수행하는 결과를 볼 수 있다.

**3.2**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**4. 함수 kill, pause**

**4.1**

**#include <unistd.h>**

**#include <signal.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**int pid;**

**void sigalarm\_handler() {**

**exit(0);**

**}**

**void sigusr\_handler() {**

**if (pid==0)**

**printf("child...\n");**

**else**

**printf("parent...\n");**

**}**

**int main(void)**

**{**

**if(signal(SIGUSR1, sigusr\_handler) == SIG\_ERR) {**

**perror("signal");**

**exit(1);**

**}**

**if(signal(SIGALRM, sigalarm\_handler) == SIG\_ERR) {**

**perror("signal");**

**exit(1);**

**}**

**pid = fork();**

**if (pid < 0) {**

**perror("fork");**

**exit(1);**

**}**

**else if (pid == 0) {**

**alarm((unsigned int)10);**

**for(;;) {**

**sleep(1);**

**kill(getppid(), SIGUSR1);**

**pause();**

**}**

**}**

**else {**

**alarm((unsigned int)10);**

**for(;;) {**

**pause();**

**sleep(1);**

**kill(pid, SIGUSR1);**

**}**

**}**

**}**

4번 함수 kill, pause는 parent가 child를 생성한 후, 매초 마다 parent와 child가 서로 번갈아 가면서 상대방에게 신 호 SIGUSR1을 보내는 프로그램이다. 두 프로세스는 10초 동안만 수행한 후 종료하 여야 하고, 두 프로세스는 signal을 받았을 때 parent는 "parent...\n"를 child는 "child...\n"를 표준 출력에 출력하여야 한다. 이 프로그램 작성을 위하여 시스템 호출 signal, kill, pause, alarm, sleep을 사용하여야 한다.

먼저 여러 헤더파일들을 include해야 하는데, sleep(), alarm(), fork(), getppid(), pause()함수를 사용하기 위해 <unistd.h>를, exit() 함수를 사용하기 위해 <stdlib.h>를, printf()함수를 사용하기 위해 <stdio.h>를, 그리고, signal(), kill()함수를 사용하기 위해 <signal.h>를 include 했다.

다음 parent process에서 child process를 만들어야 하기 때문에 fork의 return value를 저장하기 위한 int 타입의 pid라는 변수를 정의해주었다. fork의 return값은 child는 0, parent는 child pid이고 error 시 -1을 반환한다. 그리고 child 프로세스를 생성하기 위하여 fork 시스템 호출을 사용했다. fork()를 부르는 순간 parent 프로세스의 address space를 복사하여 child 프로세스를 생성한다. fork()는 한번 call 되지만 두 곳에서 return이된다. 생성 도중 error가 발생한다면 -1이 return이 되기 때문에 if문을 사용하여 pid가 0보다 작을 경우 perror(“fork”)를 호출하여 에러 메시지를 출력하고 exit(1)함수를 호출하여 비정상적인 종료했다.

만약 pid가 0인 경우는 child process가 실행하는 부분이고, pid가 0이 아니라면 parents process가 실행하는 부분이다.

이렇게 생성된 두 process(parent, child)는 매초마다 서로 번갈아 가며 상대방에게 SIGUSR1 signal을 보내야 한다. 주어진 process에게 signal을 보내기 위해 kill()함수를 사용했고, 매초 서로 번갈아 가며 상대방에게 signal을 보내야 하기 때문에 pause()함수를 사용하였다.

우선 각 process는 계속해서 신호를 보내고 있어야 되기 때문에 두 process모두 for(;;) 무한 루프속에서 수행시켰다. Kill()함수를 사용하여 다른 프로세스에 signal을 보내는데 kill()함수의 첫 번째 인수에는 signal을 전달받을 process의 pid를 주어야 하기 때문에 (pid==0)child process에서는 kill()의 첫 번째 인수로 getppid()함수를 사용하여 parent process의 pid를 주었고 parent proess의 kill()의 첫 번째 인수에는 fork()함수의 return값이 저장된 child process의 pid인 pid전역변수를 주었다. 두 process의 kill()의 두 번째 인수로 SIGUSR1 signal을 주어 두 process가 서로 상대방에서 signal을 보내도록 하였다.

이 때 두 process는 번갈아 가며 signal를 보내야 하기 때문에 두 process모두 pause()함수를 사용하여 각 process가 signal를 받을 때까지 기다리게 하였고, sleep(1)함수를 사용하여 signal을 받아 표준 출력에 “parent…\n” 및 “child…\n”를 출력한 후에 1초 동안 기다리도록 하였다.

이 때 함수를 호출하는 순서가 중요한데 모범 수행 화면과 같이 “parent…\n”이 먼저 출력되고 그 다음 “child…\n”이 출력된 뒤 번갈아 가며 출력을 하다 “parent…\n”로 출력을 끝내며 출력물의 개수가 9개 이기 때문에 먼저 1초를 기다린 뒤 parent process에서 출력을 시작했음을 알 수 있다. 따라서 child process는 for(;;) 루프를 시작하자마자 sleep(1)을 한 후에, kill()함수를 사용해 parent에게 signal을 보내 parent process에서 먼저 출력할 수 있도록 하였고 그 후 pause()함수를 호출하여 parent process에서 signal이 오는 것을 기다리도록 작성 하였고, parent process는 for(;;)루프를 시작하자 마자 pause()함수를 호출하여 signal을 기다리다 signal을 받고 “parent…\n”를 출력한 뒤 sleep(1)하고, 다시 child process에게 signal을 보내 child process가 “child…\n”를 출력하도록 하였다.

이 프로그램의 두 process는 10초 동안만 수행한 후 종료하 여야 하고, 두 프로세스는 signal을 받았을 때 parent는 "parent...\n"를 child는 "child...\n"를 표준 출력에 출력하여야 한다.

10초 동안만 수행하기 위해 alarm()함수를 사용했다. 함수 alarm()은 주어진 시간(초)가 지난 후 SIGALRM이 자신의 process에게 전달되게 한다. Alarm()함수의 첫 번째 인수는 unsigned int형 이므로 (unsigned int)10을 인수로 주어 10초 뒤에 SIGALRM을 자신의 process에게 전달하도록 parent process와 child process의 시작에 alarm()함수를 호출했다.

alarm()함수에 의해 SIGALRM signal이 process에 전달되면 프로세스가 종료할 수 있도록 signal()함수에 첫 번째 인수로 SIGALRM를 주고 두 번째 인수로 sigalarm\_handler를 주어 호출했다. 이 때 Signal()함수는 SIGALRM signal을 받으면 sigalarm\_handler()함수를 수행하는데 sigalarm\_handler()함수는 exit(0)을 수행하여 프로그램을 종료하도록 했다. 이 때 만약 signal()함수의 return값이 SIG\_ERR라면 perror()함수를 사용하여 오류 메시지를 출력하고 exit(1)함수를 사용하여 프로그램을 오류로 종료했다. 따라서 10초뒤 alarm()함수에 의해 SIGALRM signal이 process에게 전달되면 signal()함수에 의해 sigalarm\_handler()가 수행되고 이 함수는 exit(0)을 수행하므로 프로그램이 종료하게 된다.

마지막으로 두 process는 kill()함수를 사용하여 상대 process에게 SIGUSR1 signal을 보내는데 두process는 이 SIGUSR1 signal을 받으면 parent process는 “parent…\n”를 child process는 “child…\n”을 출력해야 한다. 따라서 main()함수를 시작하고 signal()함수에 첫 번째 인수로 SIGUSR1을 주고, 두 번째 인수로 sigusr\_handler를 주어 호출했다. 이 때 Signal()함수는 SIGUSR1 signal을 받으면 sigusr\_handler()함수를 수행하는데 sigusr\_handler()함수는 만약 pid가 0이라면(child process가 SIGUSR1 signal을 catch하여 호출한 경우) “child…\n”를 printf()함수를 사용하여 출력하고, pid가 양수라면(parent process가 SIGUSR1 signal을 catch하여 호출한 경우) “parent…\n”를 printf()함수를 사용하여 출력한다. 이 때 pid는 전역 변수로 main함수 안에서 fork()함수를 호출하여 return값이 저장된다.

다음과 같이 프로그래밍 한 후 명령 줄 "ps; date; ./pause ; date; ps”를 수행하면 모범 수행 화면과 결과가 같음을 알 수 있다.

**4.2**

**텍스트, 영수증, 스크린샷이(가) 표시된 사진

자동 생성된 설명**