컴퓨터학부 20201841 박세연

<wait1>

1. 소스코드

**[ssu\_wait\_1.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

void ssu\_echo\_exit(int status);

int main(void) {

pid\_t pid;

int status;

// 첫 번째 자식 프로세스 생성

if ((pid = fork()) < 0) { // fork 실패 시

fprintf(stderr, "fork error\n");

exit(1);

} else if (pid == 0) { // 자식 프로세스에서 실행할 코드

exit(7); // 종료 상태를 7로 설정하고 종료

}

// 첫 번째 자식 프로세스의 종료를 기다림

if (wait(&status) != pid) { // 자식 프로세스의 종료 상태를 기다리고 그 PID가 반환되지 않을 경우

fprintf(stderr, "wait error\n");

exit(1);

}

// 종료 상태 출력 함수 호출

ssu\_echo\_exit(status);

// 두 번째 자식 프로세스 생성

if ((pid = fork()) < 0) { // fork 실패 시

fprintf(stderr, "fork error\n");

exit(1);

} else if (pid == 0) { // 자식 프로세스에서 실행할 코드

abort(); // 프로세스를 비정상적으로 종료

}

// 두 번째 자식 프로세스의 종료를 기다림

if (wait(&status) != pid) { // 자식 프로세스의 종료 상태를 기다리고 그 PID가 반환되지 않을 경우

fprintf(stderr, "wait error\n");

exit(1);

}

// 종료 상태 출력 함수 호출

ssu\_echo\_exit(status);

// 세 번째 자식 프로세스 생성

if ((pid = fork()) < 0) { // fork 실패 시

fprintf(stderr, "fork error\n");

exit(1);

} else if (pid == 0) { // 자식 프로세스에서 실행할 코드

status /= 0; // 0으로 나누어 예외 발생

}

// 세 번째 자식 프로세스의 종료를 기다림

if (wait(&status) != pid) { // 자식 프로세스의 종료 상태를 기다리고 그 PID가 반환되지 않을 경우

fprintf(stderr, "wait error\n");

exit(1);

}

// 종료 상태 출력 함수 호출

ssu\_echo\_exit(status);

exit(0); // 프로그램 종료

}

// 종료 상태를 출력하는 함수 정의

void ssu\_echo\_exit(int status) {

if (WIFEXITED(status)) // 정상 종료인 경우

printf("normal termination, exit status = %d\n", WEXITSTATUS(status)); // 종료 상태 출력

else if (WIFSIGNALED(status)) // 시그널에 의한 종료인 경우

printf("abnormal termination, signal number = %d%s\n", WTERMSIG(status),

#ifdef WCOREDUMP

WCOREDUMP(status) ? " (core file generated)" : ""); // 코어 덤프 여부 출력

#else

"");

#endif

else if (WIFSTOPPED(status)) // 중지된 경우

printf("child stopped, signal number = %d\n", WSTOPSIG(status)); // 중지된 시그널 번호 출력

}

1. 실행결과

텍스트, 소프트웨어, 멀티미디어 소프트웨어, 스크린샷이(가) 표시된 사진

자동 생성된 설명

<wait2>

1. 소스코드

**[ssu\_wait\_2.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#define EXIT\_CODE 1

int main(void) {

pid\_t pid;

int ret\_val, status;

// 자식 프로세스 생성

if ((pid = fork()) == 0) { // 자식 프로세스에서는 fork()의 반환값이 0이므로 if문 내부 실행

printf("child: pid = %d ppid = %d exit\_code = %d\n", getpid(), getppid(), EXIT\_CODE); // 자식 프로세스에서 출력: 자식의 PID, 부모의 PID, 종료 코드 출력

exit(EXIT\_CODE); // 자식 프로세스 종료

}

printf("parent: waiting for child = %d\n", pid); // 부모 프로세스에서 출력: 자식 프로세스의 PID 대기 중

ret\_val = wait(&status); // 부모 프로세스는 자식 프로세스의 종료를 기다림

printf("parent: return value = %d, ", ret\_val); // 부모 프로세스에서 출력: wait 함수의 반환값 출력

printf("child's status = %x", status); // 부모 프로세스에서 출력: 자식 프로세스의 종료 상태 출력

printf(" and shifted = %x\n", (status >> 8)); // 부모 프로세스에서 출력: 자식 프로세스의 종료 상태를 비트 시프트하여 출력

exit(0); // 프로그램 종료

}

2. 실행결과

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

<wait3>

1. 소스코드

**[ssu\_wait\_3.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

int main(void) {

// 첫 번째 자식 프로세스 생성 후 "echo" 명령어 실행

if (fork() == 0)

execl("/bin/echo", "echo", "this is", "message one", (char \*)0);

// 두 번째 자식 프로세스 생성 후 "echo" 명령어 실행

if (fork() == 0)

execl("/bin/echo", "echo", "this is", "message Two", (char \*)0);

printf("parent: waiting for children\n"); // 부모 프로세스에서 출력: 자식 프로세스 종료 대기 중

// 자식 프로세스의 종료를 기다림

while(wait((int\*)0) != -1);

printf("parent: all children terminated\n"); // 부모 프로세스에서 출력: 모든 자식 프로세스 종료됨

exit(0); // 프로그램 종료

}

2. 실행결과

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

<wait4>

1. 소스코드

**[ssu\_wait\_4.c]**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

int main(void) {

pid\_t child1, child2; // 변수 선언: 자식 프로세스의 PID를 저장할 변수

int pid, status; // 변수 선언: 자식 프로세스의 종료 상태를 저장할 변수

// 첫 번째 자식 프로세스 생성

if ((child1 = fork()) == 0) // 자식 프로세스에서는 fork()의 반환값이 0이므로 if문 내부 실행

execlp("date", "date", (char\*)0); // date 명령어 실행

// 두 번째 자식 프로세스 생성

if ((child2 = fork()) == 0) // 자식 프로세스에서는 fork()의 반환값이 0이므로 if문 내부 실행

execlp("who", "who", (char\*)0); // who 명령어 실행

printf("parent: waiting for children\n"); // 부모 프로세스에서 출력: 자식 프로세스 종료 대기 중

// 자식 프로세스의 종료를 기다림

while ((pid = wait(&status)) != -1) {

if (child1 == pid) // 첫 번째 자식 프로세스 종료 시

printf("parent: first child: %d\n", (status >> 8)); // 종료 상태 출력

else if (child2 == pid) // 두 번째 자식 프로세스 종료 시

printf("parent: second child %d\n", (status >> 8)); // 종료 상태 출력

}

printf("parent: all children terminated\n"); // 부모 프로세스에서 출력: 모든 자식 프로세스 종료됨

exit(0); // 프로그램 종료

}

2. 실행결과

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명