컴퓨터학부 20202920 조민혁

1.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;

if((pid = fork()) < 0){

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

exit(1);

}//fork를 수행시켜 줍니다.

else if(pid == 0)

exit(7);

//자식 프로세스에서 수행시켜줍니다.

if(wait(&status) != pid){

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

exit(1);

}//wait를 통해 자식프로세스가 죽고 난 후의 상태를 저장합니다.

ssu\_echo\_exit(status);

//ssu\_echo\_exit함수를 수행시켜줍니다.

if((pid = fork()) < 0){

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

exit(1);

}//fork시켜줍니다.

else if(pid == 0)

abort();

//abort함수를 수행시켜줍니다.

if(wait(&status) != pid){

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

exit(1);

}//wait함수를 수행시켜줍니다.

ssu\_echo\_exit(status);

if((pid = fork()) < 0){

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

exit(1);

}

//fork 시켜줍니다.

else if(pid == 0)

status /= 0;

if(wait(&status) != pid){

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

exit(1);

}//wait함수를 수행시켜줍니다.

ssu\_echo\_exit(status);

exit(0);

}

void ssu\_echo\_exit(int status){

//ssu\_echo\_exit함수를 정의해줍니다.

//매크로들을 통해 상태를 판별합니다.

if(WIFEXITED(status))

//자식 프로세스가 정상적으로 종료되었는지 판단합니다.

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

else if(WIFSIGNALED(status))

printf("abnoraml termination, siganl number = %d%s\n",

WTERMSIG(status),

#ifdef WCOREDUMP

WCOREDUMP(status) ? " (core file generated)" : "");

#else

"");

#endif

else if (WIFSTOPPED(status))

printf("child stopped, signal number = %d\n",WSTOPSIG(status));

}

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

2.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;

//변수를 선언해줍니다.

if((pid = fork()) == 0){

printf("childe: pid = %d ppid = %d exit\_code = %d\n",

getpid(),getppid(),EXIT\_CODE);

exit(EXIT\_CODE);

}//fork를 생성시켜주고 자식프로세스의 경우 출력시켜줍니다.

printf("parent: waiting for child = %d\n",pid);

ret\_val = wait(&status);

//성공한 프로세스ID를 저장시켜줍니다.

printf("parent: return value = %d, ", ret\_val);

printf(" child's status = %x",status);

printf(" and shifted = %x\n", (status >> 8));

//printf로 출력시켜줍니다.

exit(0);

}

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

3.ssu\_wait\_3.c

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

int main(void)

{

if(fork() == 0)

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

//자식프로세스의 경우 execl함수를 수행시킵니다.

if(fork() == 0)

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

printf("parent: waiting for children\n");

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

//부모 프로세스가 모든 자식 프로세스의 종료를 기다립니다.

printf("parent: all children terminated\n");

exit(0);

}

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

4.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;

int pid, status;

if((child1 = fork()) == 0)

execlp("date", "date", (char \*)0);

//fork()시켜주고 exec함수를 수행시켜줍니다.

if((child2 = fork()) == 0)

execlp("who", "who", (char \*)0);

//두번째 fork시켜줍니다.

printf("parent: waiting for children\n");

//자식의 wait을 통해 얻은 status를 수행시켜줍니다.

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

if (child1 == pid)

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

else if (child2 == pid)

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

}

printf("parent: all children terminated\n");

exit(0);

}

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명