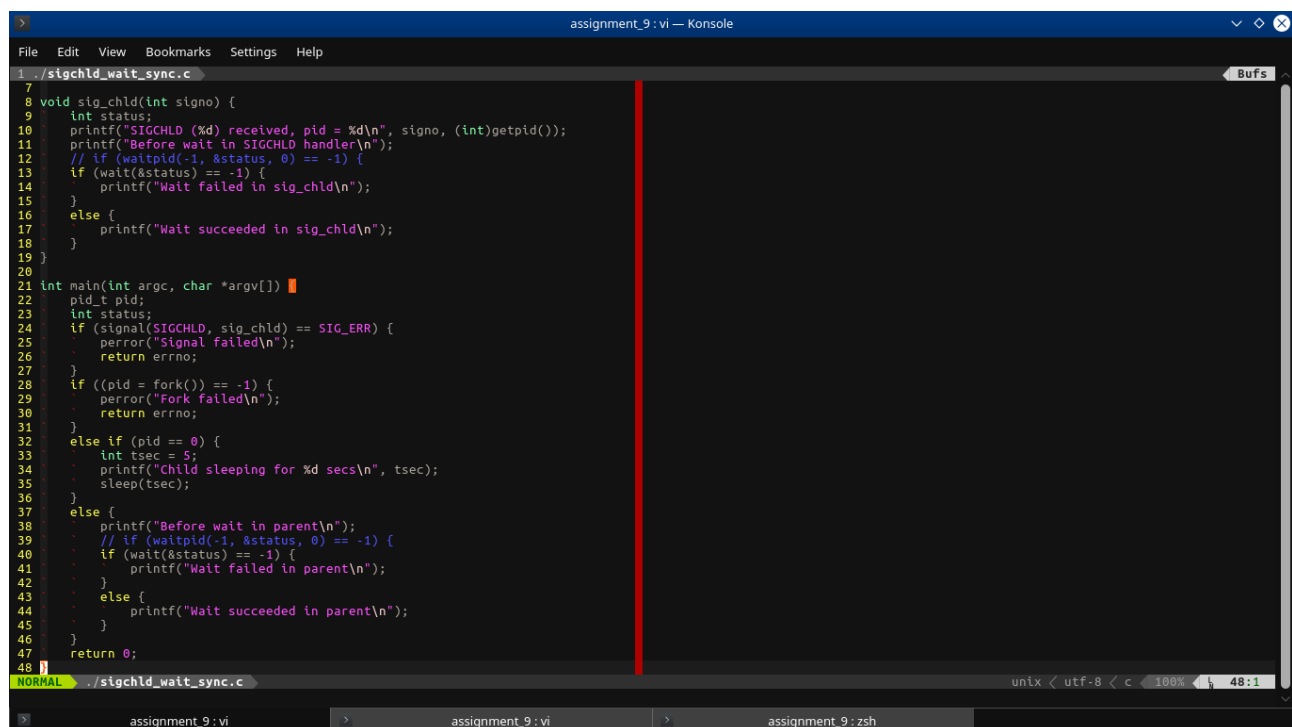


Assignment 9

Niramay Vaidya 111605075
Srishti Shelke 111603056

1. Write a program to understand the proper functioning of SIGCHLD and wait () synchronization. Create a signal handler for SIGCHLD and in the handler, call wait. In the handler, keep a message before wait and another message for the success or failure of wait. In the main function, fork and let the parent wait for the child. In the main function, keep a message before wait and another message for the success or failure of wait. Elaborate your understanding.

Code-



```
1 ./sigchld_wait_sync.c
7
8 void sig_chld(int signo) {
9     int status;
10    printf("SIGCHLD (%d) received, pid = %d\n", signo, (int)getpid());
11    printf("Before wait in SIGCHLD handler\n");
12    // if (waitpid(-1, &status, 0) == -1) {
13    if (wait(&status) == -1) {
14        printf("Wait failed in sig_chld\n");
15    }
16    else {
17        printf("Wait succeeded in sig_chld\n");
18    }
19 }
20
21 int main(int argc, char *argv[]) {
22     pid_t pid;
23     int status;
24     if (signal(SIGCHLD, sig_chld) == SIG_ERR) {
25         perror("Signal failed\n");
26         return errno;
27     }
28     if ((pid = fork()) == -1) {
29         perror("Fork failed\n");
30         return errno;
31     }
32     else if (pid == 0) {
33         int tsec = 5;
34         printf("Child sleeping for %d secs\n", tsec);
35         sleep(tsec);
36     }
37     else {
38         printf("Before wait in parent\n");
39         // if (waitpid(-1, &status, 0) == -1) {
40         if (wait(&status) == -1) {
41             printf("Wait failed in parent\n");
42         }
43         else {
44             printf("Wait succeeded in parent\n");
45         }
46     }
47     return 0;
48 }
```

Output-



```
inramay@itransy: ~/Documents/sem7/AUP/assignment_9: ./sigchld_wait_sync
Before wait in parent
Child sleeping for 5 secs
SIGCHLD (17) received, pid = 3375
Before wait in SIGCHLD handler
Waitpid failed in sig_chld
Waitpid succeeded in parent
```

2. Write a program to understand the proper functioning of blocked SIGCHLD and wait(). In the main function, block SIGCHLD and fork(). Let the child exits immediately and parent sleep(5), display the pending signals and wait for the child. Is there any need for SIGCHLD to be delivered for the wait() to return? Elaborate your understanding.

Code-

```
assignment_9: vi — Konsole
File Edit View Bookmarks Settings Help
1 ./sigchld_block_wait.c
9 void sig_chld(int signo) {
10     printf("SIGCHLD (%d) received, pid = %d\n", signo, (int)getpid());
11 }
12
13 int main(int argc, char *argv[]) {
14     pid_t pid;
15     int status, err;
16     sigset_t sigset, sigsetpending;
17     if (signal(SIGCHLD, sig_chld) == SIG_ERR) {
18         perror("Signal failed\n");
19         return errno;
20     }
21     if (sigemptyset(&sigset) == -1) {
22         perror("Sigemptyset failed\n");
23         return errno;
24     }
25     if (sigaddset(&sigset, SIGCHLD) == -1) {
26         perror("Sigaddset for SIGCHLD failed\n");
27         return errno;
28     }
29     if (sigprocmask(SIG_SETMASK, &sigset, NULL) == -1) {
30         perror("Sigprocmask failed\n");
31     }
32     if ((pid = fork()) == -1) {
33         perror("Fork failed\n");
34         return errno;
35     }
36     else if (pid == 0) {
37         exit(0);
38     }
39     else {
40         printf("Parent sleeping for 5 secs\n");
41         sleep(5);
42         if (sigpending(&sigsetpending) == -1) {
43             perror("Sigpending failed\n");
44             return errno;
45         }
46     }
47 }
NORMAL ./sigchld_block_wait.c
unix < utf-8 < c 37% 23:21
```

```
assignment_9: vi — Konsole
File Edit View Bookmarks Settings Help
1 ./sigchld_block_wait.c
45 }
46 if ((err = sigismember(&sigsetpending, SIGCHLD)) == -1) {
47     perror("Sigismember failed\n");
48     return errno;
49 }
50 else if (err == 1) {
51     printf("SIGCHLD pending\n");
52 }
53 // if (waitpid(-1, &status, 0) == -1) {
54 if (wait(&status) == -1) {
55     perror("Wait failed\n");
56     return errno;
57 }
58 else {
59     printf("Wait returned\n");
60 }
61 return 0;
62 }
63 }
```

Output-

```
niranmay@niranmay: ~/Documents/sem7/AUP/assignment_9 ./sigchld_block_wait
Parent sleeping for 5 secs
SIGCHLD pending
Waitpid returned
niranmay@niranmay: ~/Documents/sem7/AUP/assignment_9
```

Explanation-

Providing a combined explanation for both the questions since individual explanations for both are dependent on a common source of information.

(Advanced Programming in the UNIX Environment - Stevens)

```
#include "apue.h"

static void
sig_int(int signo)
{
    printf("caught SIGINT\n");
}

static void
sig_chld(int signo)
{
    printf("caught SIGCHLD\n");
}

int
main(void)
{
    if (signal(SIGINT, sig_int) == SIG_ERR)
        err_sys("signal(SIGINT) error");
    if (signal(SIGCHLD, sig_chld) == SIG_ERR)
        err_sys("signal(SIGCHLD) error");
    if (system("/bin/ed") < 0)
        err_sys("system() error");
    exit(0);
}
```

Figure 10.26 Using system to invoke the ed editor

```

if (sigprocmask(SIG_BLOCK, &chldmask, &savemask) < 0)
    return(-1);

if ((pid = fork()) < 0) {
    status = -1; /* probably out of processes */
} else if (pid == 0) { /* child */
    /* restore previous signal actions & reset signal mask */
    sigaction(SIGINT, &saveintr, NULL);
    sigaction(SIGQUIT, &savequit, NULL);
    sigprocmask(SIG_SETMASK, &savemask, NULL);

    execl("/bin/sh", "sh", "-c", cmdstring, (char *)0);
    _exit(127); /* exec error */
} else { /* parent */
    while (waitpid(pid, &status, 0) < 0)
        if (errno != EINTR) {
            status = -1; /* error other than EINTR from waitpid() */
            break;
        }
}

/* restore previous signal actions & reset signal mask */
if (sigaction(SIGINT, &saveintr, NULL) < 0)
    return(-1);
if (sigaction(SIGQUIT, &savequit, NULL) < 0)
    return(-1);
if (sigprocmask(SIG_SETMASK, &savemask, NULL) < 0)
    return(-1);

return(status);
}

```

Figure 10.28 Correct POSIX.1 implementation of system function

If we link the program in Figure 10.26 with this implementation of the system function, the resulting binary differs from the last (flawed) one in the following ways.

1. No signal is sent to the calling process when we type the interrupt or quit character.
2. When the ed command exits, SIGCHLD is not sent to the calling process. Instead, it is blocked until we unblock it in the last call to sigprocmask, after the system function retrieves the child's termination status by calling waitpid.

POSIX.1 states that if wait or waitpid returns the status of a child process while SIGCHLD is pending, then SIGCHLD should not be delivered to the process unless the status of another child process is also available. None of the four implementations discussed in this book implements this semantic. Instead, SIGCHLD remains pending after the system function calls waitpid; when the signal is unblocked, it is delivered to the caller. If we called wait in the sig_chld function in Figure 10.26, it would return -1 with errno set to ECHILD, since the system function already retrieved the termination status of the child.

In response to the first question, the above stated information says that in the implementation of system, if wait had been called in the signal handler for SIGCHLD set by the caller as a disposition, it would have returned -1 since when waitpid was called within system, it had already retrieved the termination status of the child. Similarly, in the code image for the first question, when wait was called in the parent, it retrieved the termination status of the child, and hence, when wait was again called in the signal handler for SIGCHLD, it returned -1. When wait is called in the parent and subsequently blocks the parent, even though another wait is called in the SIGCHLD handler, the termination status has been collected by the previous wait immediately after the generation of SIGCHLD. Hence, even if the handler code executes first, wait in it will return -1, since the wait in

parent has already collected the termination status of its child, has removed parent out of the blocking state. The parent continues after the execution of the handler.

In response to the second question, the above stated information says in an indirect way that there is no dependency of wait returning on the delivery of SIGCHLD if this signal has been blocked and is pending. Wait will return by collecting the termination status of its child once the child has finished executing. A SIGCHLD will have been generated upon the termination of the child, but if the signal is blocked in the parent, its delivery remains pending but does not obstruct wait from returning to the parent. When later on in the parent, if the SIGCHLD is unblocked, it is delivered to the parent.