AUP : Assignment - 8 [Advanced Signals]

Aditi Rajendra Medhane 111803177

25th October 2021

Q1

"Child inherit parent's signal mask when it is created, but pending signals for the parent process are not passed on". Write appropriate program and test with suitable inputs to verify this.

Code

```c
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>
#include <errno.h>
#include <stdio.h>
#include <wait.h>

void print_message(int signo) {
        printf("%d hit by signal %d\n", getpid(), signo);
}


int main(void) {
        struct sigaction sigint_handler;
        sigset_t old, new;

        sigint_handler.sa_handler = print_message;
        sigemptyset(&(sigint_handler.sa_mask));

        /* read current mask */
        if (sigprocmask(0, NULL, &new) == -1) {
                perror("sigprocmask- read");
                return errno;
        }

        if (sigaction(SIGINT, &sigint_handler, NULL) == -1) {
                perror("sigaction");
                return errno;
        }

        /* block SIGINT */
        if (sigaddset(&new, SIGINT) == -1) {
                perror("sigaddset");
                return errno;
        }

        if (sigprocmask(SIG_BLOCK, &new, &old) == -1) {
                perror("sigprocmask- modify");
                return errno;
        }

        printf("Press any key to continue\n");

        getchar();

        int child_pid, status, wait_ret;
```

```
47          if ((child_pid = fork()) == -1) {
48                  perror("fork");
49                  return errno;
50          }
51          else if (child_pid) {
52                  /* parent */
53                  /* stop blocking SIGINT */
54                  printf("Parent: %d\n", getpid());
55                  if (sigprocmask(SIG_SETMASK, &old, NULL) == -1) {
56                          perror("sigprocmask- reset old, parent");
57                          return errno;
58                  }
59
60                  if ((wait_ret = wait(&status)) == -1) {
61                          perror("wait");
62                          return errno;
63                  }
64
65                  printf("Child %d returned\n", wait_ret);
66
67          }
68          else {
69                  /* child */
70                  printf("Child: %d\n", getpid());
71                  /* stop blocking SIGINT */
72                  if (sigprocmask(SIG_SETMASK, &old, NULL) == -1) {
73                          perror("sigprocmask- reset old, child");
74                          return errno;
75                  }
76          }
77
78          return 0;
79  }
```

Explanation

- SIGINT is blocked in **parent**, and a handler for SIGINT is *set*.
- During execution, send SIGINT to parent from terminal before it forks. This means that *SIGINT is not pending*.
- After forking, SIGINT is unblocked.
- It is observed that **SIGINT hits only the parent, and not the child**.

Output

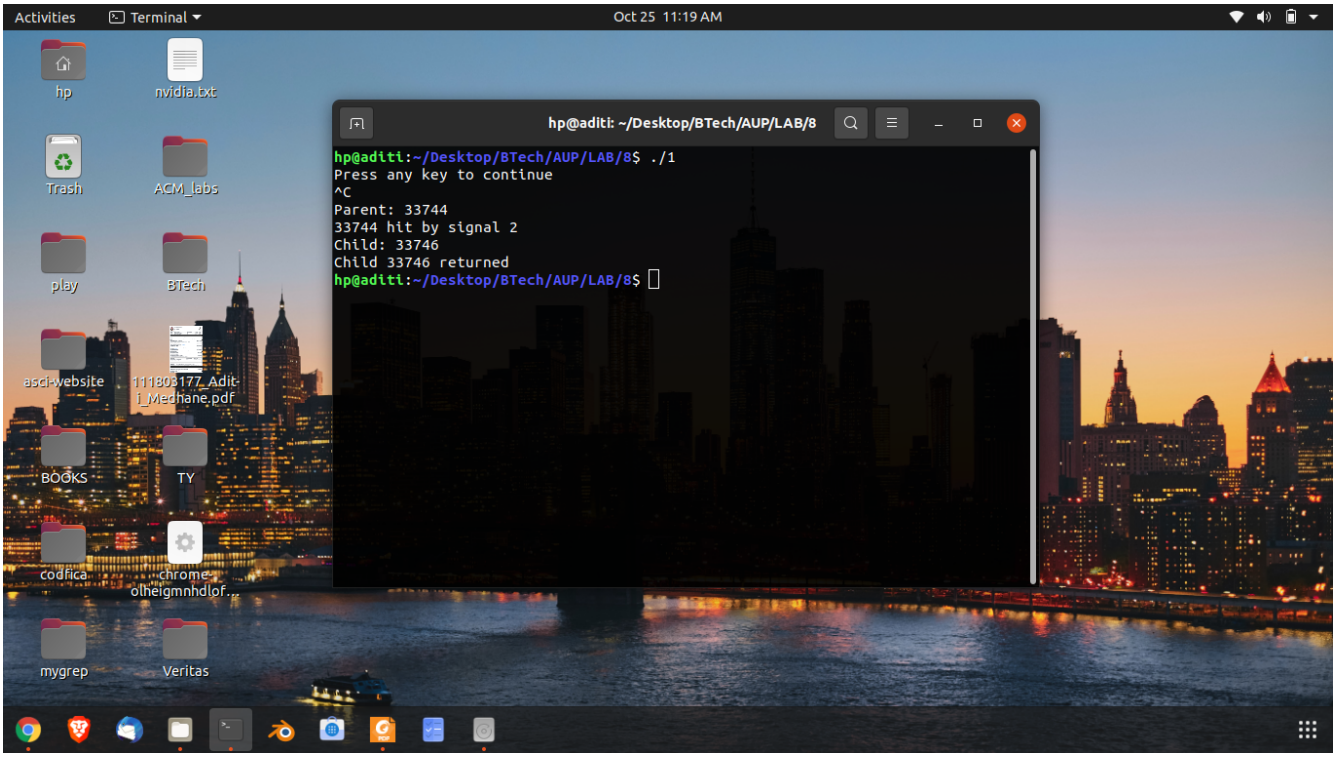

Figure 1: Output : When unblocked after fork(), SIGINT only hits parent

Q2

Write a program to understand the proper functioning of SIGCHLD and wait () synchronization. Create a signal handler for SIGCHILD 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 about which wait collects the termination status of the child. What is the sequence of execution of wait and signal handler?

Code

```c
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>
#include <errno.h>
#include <stdio.h>
#include <wait.h>

void reaper(int signo) {
        int status, wait_ret;
        printf("Waiting in signal handler\n");
        if ((wait_ret = wait(&status)) == -1) {
                printf("Wait in signal handler failed\n");
        }
        else {
                printf("Wait in signal handler returned %d\n", wait_ret);
        }
}


int main(void) {
        struct sigaction sigint_handler;

        sigint_handler.sa_handler = reaper;
        sigemptyset(&(sigint_handler.sa_mask));

        if (sigaction(SIGCHLD, &sigint_handler, NULL) == -1) {
                perror("sigaction");
                return errno;
```

```
        }

        int child_pid, status, wait_ret;
        if ((child_pid = fork()) == -1) {
                perror("fork");
                return errno;
        }
        else if (child_pid) {
                /* parent */
                printf("Parent: %d\n", getpid());
                printf("Waiting in parent, after fork\n");
                if ((wait_ret = wait(&status)) == -1) {
                        printf("Wait in parent after fork failed\n");
                }
                else {
                        printf("Wait in parent after fork returned %d\n", wait_ret);
                }

        }
        else {
                /* child */
                printf("Child: %d\n", getpid());
                return 0;
        }

        return 0;
}
```

Explanation

- The wait() called after fork **collects the termination status of the child**.
- If wait() is blocking before the child exits, then the wait returns first. Immediately after wait() returns, SIGCHLD hits and it's handler runs.
- Because the child has already been reaped, the wait() inside the handler fails.

Output

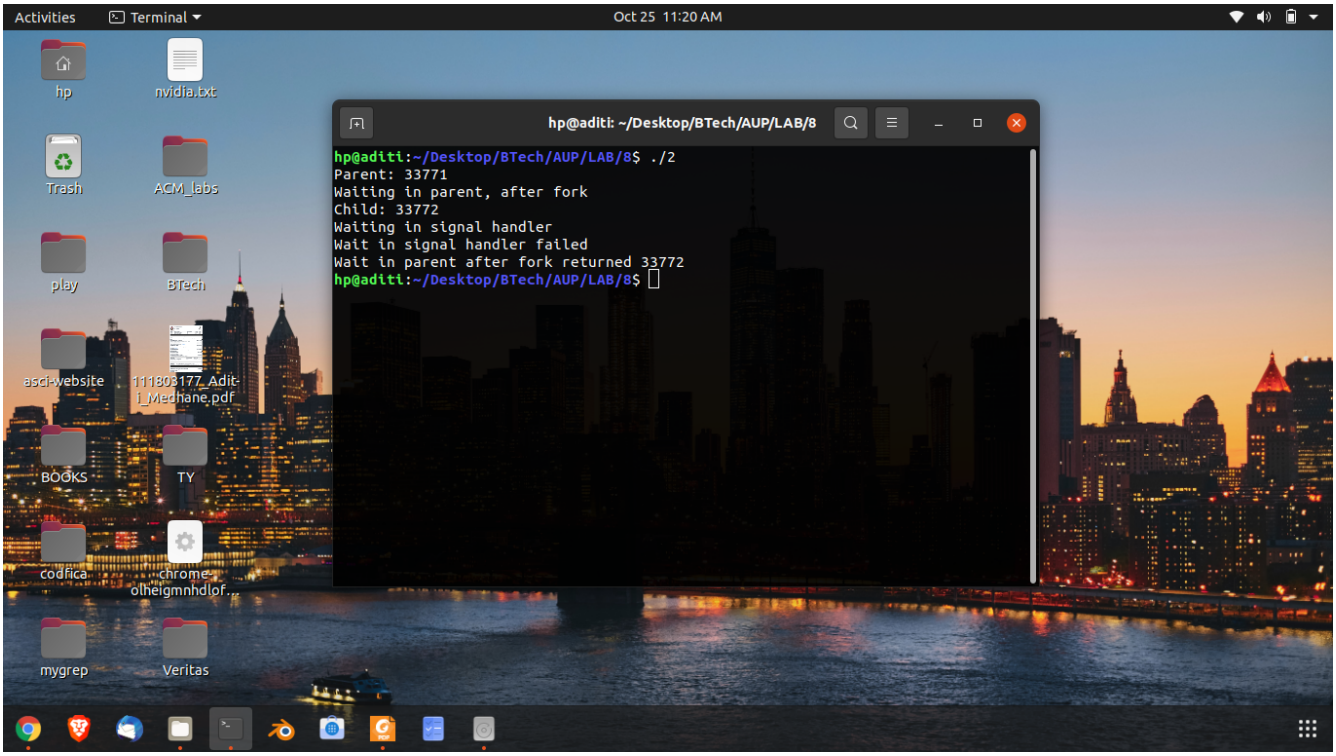

Figure 2: Output : wait() in parent returns successfully, SIGCHLD is delivered, wait() in SIGCHLD handler returns 0

Q3

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

```c
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>
#include <errno.h>
#include <stdio.h>
#include <wait.h>

int main(void) {

        sigset_t mask, pending;

        /* block SIGCHLD*/
        if (sigaddset(&mask, SIGCHLD) == -1) {
                perror("sigaddset");
                return errno;
        }

        if (sigprocmask(SIG_BLOCK, &mask, NULL) == -1) {
                perror("sigprocmask- modify");
                return errno;
        }

        int child_pid, status, wait_ret;
        if ((child_pid = fork()) == -1) {
                perror("fork");
                return errno;
        }
        else if (child_pid) {
                /* parent */
                printf("Parent: %d\n", getpid());

                sleep(5);

                if (sigpending(&pending) == -1) {
                        perror("sigpending");
                        return errno;
                }

                int is_member;
                if ((is_member = sigismember(&pending, SIGCHLD)) == -1) {
                        perror("sigismember");
                        return errno;
                }
                else if (is_member) {
                        printf("SIGCHLD is pending\n");
                }
                else {
                        printf("SIGCHLD is not pending\n");
                }

                if ((wait_ret = wait(&status)) == -1) {
                        printf("Wait in parent after fork failed\n");
                }
                else {
                        printf("Wait in parent after fork returned %d\n", wait_ret);
                }

        }
        else {
```

```
60                  /* child */
61                  printf("Child: %d\n", getpid());
62                  return 0;
63          }
64
65          return 0;
66  }
```

Explanation

- SIGCHLD remains pending when wait returns. So we can conclude that **wait() returns when child exits, not when SIGCHLD is delivered**.
- SIGCHLD does not need to be delivered for wait() to return.

Output

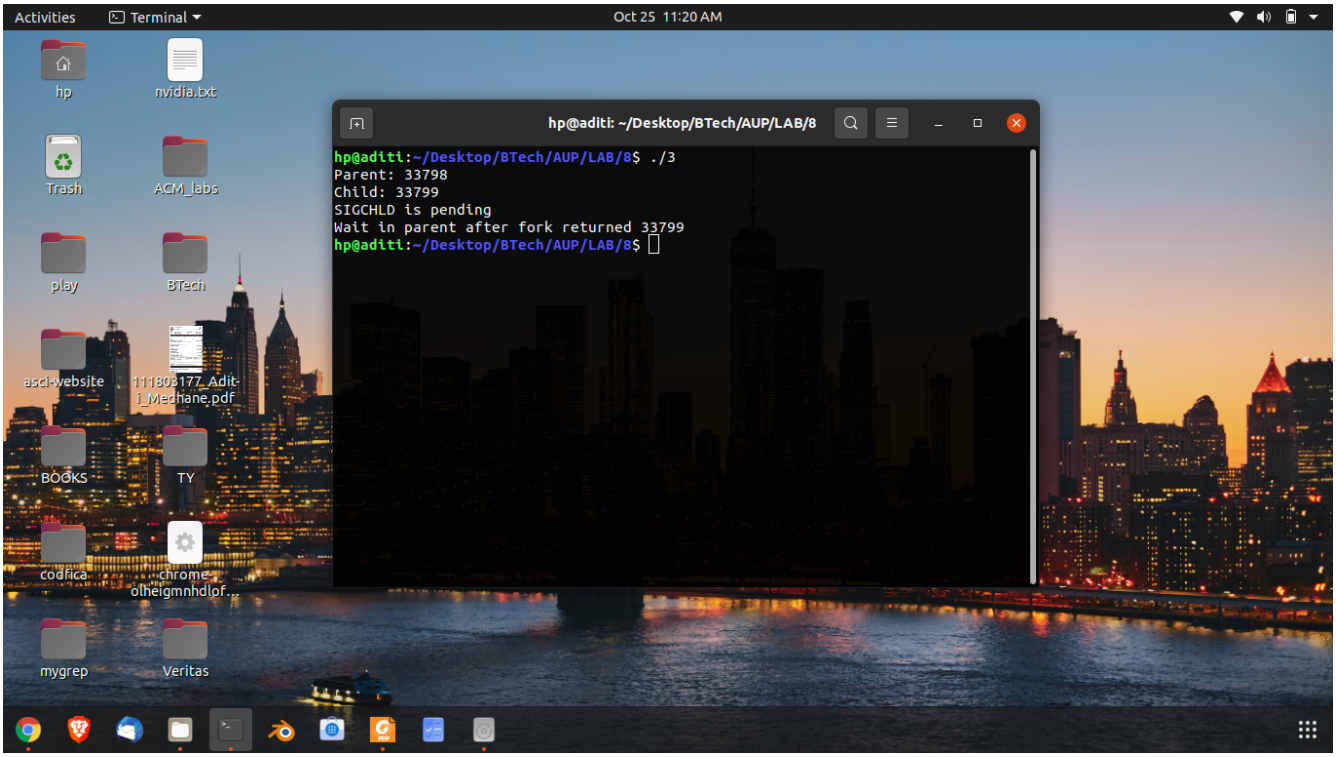

Figure 3: Output: SIGCHLD remains pending, but wait() returns