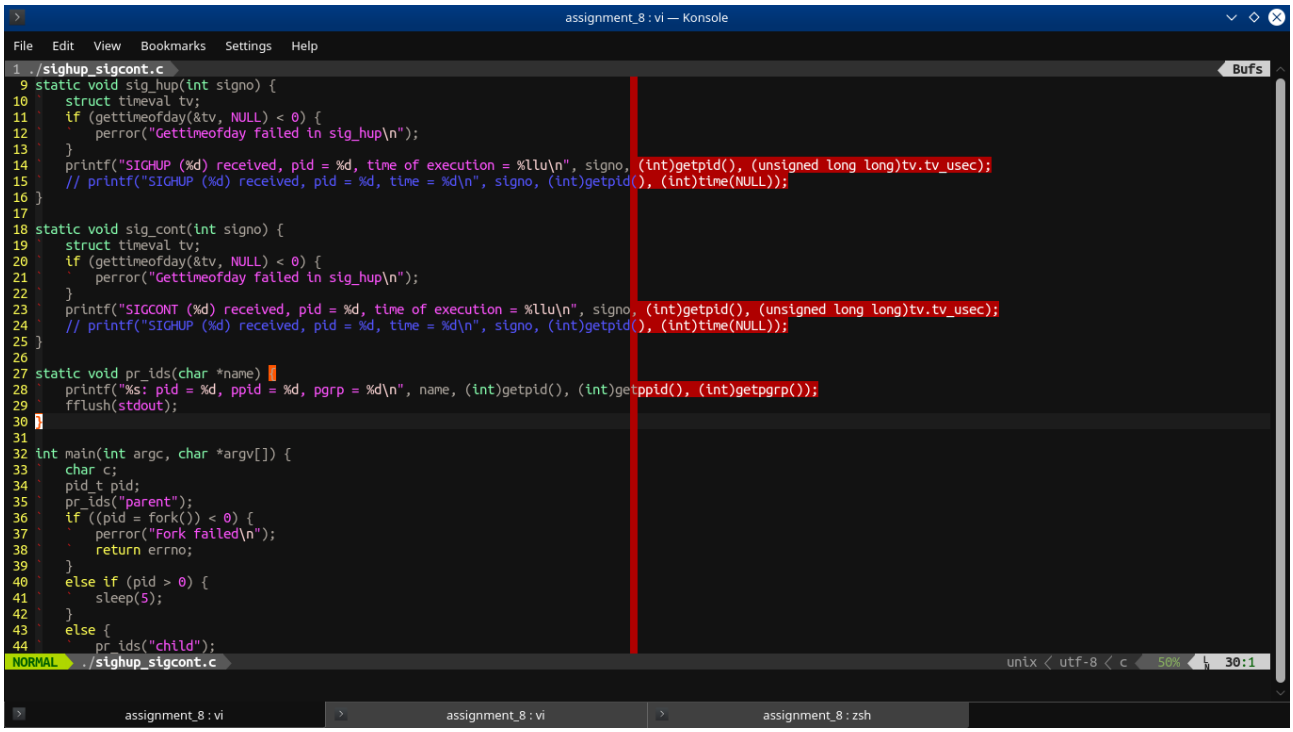
Assignment 8

Srishti Shelke 111603056
Niramay Vaidya 111605075

1. “If the termination of a process causes a process group to become orphaned, and if any member of the newly orphaned process group is stopped, then a **SIGHUP** signal followed by a **SIGCONT** signal will be sent to each process in the newly orphaned process group.” Write a program to test this feature by defining the signal handlers for SIGHUP and SIGCONT. Which signal is delivered first? Why?

Code-



```c
    if (signal(SIGHUP, sig_hup) == SIG_ERR) {
        perror("Signal for SIGHUP failed\n");
        return errno;
    }
    if (signal(SIGCONT, sig_cont) == SIG_ERR) {
        perror("Signal for SIGCONT failed\n");
        return errno;
    }
    if (kill(getpid(), SIGTSTP) < 0) {
        perror("Kill failed\n");
        return errno;
    }
    pr_ids("child");
  }
  return 0;
}
```

Output-

```
nitramay@nitramay  ~/Documents/sem7/AUP/assignment_8  cc sighup_sigcont.c -o sighup_sigcont        10026  07:41:46
nitramay@nitramay  ~/Documents/sem7/AUP/assignment_8  ./sighup_sigcont                            10026  07:43:01
parent: pid = 8542, ppid = 4859, pgrp = 8542
child: pid = 8543, ppid = 8542, pgrp = 8542
SIGCONT (18) received, pid = 8543, time = 224856
SIGHUP (1) received, pid = 8543, time = 224912
child: pid = 8543, ppid = 1, pgrp = 8542
nitramay@nitramay  ~/Documents/sem7/AUP/assignment_8  cc sighup_sigcont.c -o sighup_sigcont        10027  07:43:09
nitramay@nitramay  ~/Documents/sem7/AUP/assignment_8  ./sighup_sigcont                            10028  07:44:18
parent: pid = 8661, ppid = 4859, pgrp = 8661
child: pid = 8662, ppid = 8661, pgrp = 8661
SIGCONT (18) received, pid = 8662, time = 363094
SIGHUP (1) received, pid = 8662, time = 363151
child: pid = 8662, ppid = 1, pgrp = 8661
nitramay@nitramay  ~/Documents/sem7/AUP/assignment_8  adios                                        10029  07:44:24
nitramay@nitramay  ~/Documents/sem7/AUP/assignment_8  cc sighup_sigcont.c -o sighup_sigcont        10030  07:50:32
nitramay@nitramay  ~/Documents/sem7/AUP/assignment_8  ./sighup_sigcont                            10032  08:25:27
parent: pid = 9341, ppid = 4859, pgrp = 9341
child: pid = 9342, ppid = 9341, pgrp = 9341
SIGCONT (18) received, pid = 9342, time of execution = 300355
SIGHUP (1) received, pid = 9342, time of execution = 300415
child: pid = 9342, ppid = 1, pgrp = 9341
nitramay@nitramay  ~/Documents/sem7/AUP/assignment_8                                              10033  08:25:34
```

Explanation-

The time of execution has been printed in the sig_hup and sig_cont handlers to check which handler gets called first. According to the output above, the sig_cont handler gets called first followed by the sig_hup handler but that does not mean the SIGCONT signal is received before the SIGHUP signal.
The SIGHUP cannot be delivered until the child's execution is resumed. When a process is stopped, all signal delivery is suspended except for SIGCONT and SIGKILL. So, the SIGHUP does arrive first, but it cannot be processed until the SIGCONT awakens the process execution.

2. You have already created a process tree and a process group of (3, 4, 5) in Lab 7. Let process 0 send a signal to this group. Display appropriate messages at the sender and receiver.

Code-

assignment_8 : vi — Konsole

File Edit View Bookmarks Settings Help

./signal_process_group.c

```c
#define FIFO_NAME "transfer"

long *shared_pgid_3;

void sig_usr1(int signo) {
    printf("SIGUSR1 (%d) received, pid = %ld\n", signo, (long)getpid());
}

void sig_alrm(int signo) {
    printf("SIGALRM (%d) received, pid = %ld\n", signo, (long)getpid());
    /* this is being done to detect that the signal handler for SIGALRM has been
     * hit and hence the while loop in process 0 has run for too long since the
     * value of shared_pgid_3 which was set to -1 at the start in process 0 has
     * not yet been changed by process 5, hence make its value -2 to get out of
     * the while loop and print corresponding message by checking the value to
     * be -2
     */
    *shared_pgid_3 = -2;
}

int main(int argc, char *argv[], char *envp[]) {
    pid_t pid = 0;
    /* process 0 */
    /* calling mmap to create a shared memory region here ensures that
     * process 5 has already been moved to the process group created by
     * 3
     * the shared memory will contain the pgid of process group created
     * by 3 which will then be read by process 0 to give a signal to it
     *
     * calling mmap here to create a shared memory region so that children of
     * process 0 and their children will have read+write access to this shared
     * memory since the call returns a pointer to such an allocated memory whose
     * location remains constantly pointed to be the global shared_pgid_3
     * pointer, this global pointer gets copied down the process tree to process
     * 5 which can then store the obtained pgid_3 in it at the location pointed
     * to by shared_pgid_3
```

NORMAL ./signal_process_group.c unix utf-8 c 5% 13:1

assignment_8 : vi | assignment_8 : zsh | man : man

assignment_8 : vi — Konsole

File Edit View Bookmarks Settings Help

./signal_process_group.c

```c
     */
    if ((shared_pgid_3 = (long *)mmap(NULL, (size_t)sizeof(long), PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0)) == MAP_FAILED) {
        perror("Mmap failed\n");
        return errno;
    }
    /* this is being done to keep running a while loop for some finite amount of
     * time defined by the alarm calls in process 0 by checking the value
     * pointed to by shared_pgid_3 against -1, the while loop will exit either
     * when the alaram expires or process 5 changes the value pointed to by
     * shared_pgid_3 to anything other than -2 and -1, since these are not valid
     * process group ids
     */
    *shared_pgid_3 = -1;
    if ((pid = fork()) == -1) {
        perror("Fork failed in process 0\n");
        return errno;
    }
    else if (pid == 0) {
        /* process 1 */
        if ((pid = fork()) == -1) {
            perror("Fork failed in process 1\n");
            return errno;
        }
        else if (pid == 0) {
            /* process 5 */
            printf("process 5 pid: %ld ppid: %ld\n", (long)getpid(), (long)getppid());
            int fd;
            /* this is required in the case when the transfer FIFO file does not
             * exist in the current directory and process 5 executes before
             * process 3, then open fails, hence process 5 should remain in a
             * while loop and only exit it once process 3 has executed and
             * created the transfer FIFO in which case open here will succeed
             */
            while ((fd = open(FIFO_NAME, O_RDONLY)) == -1);
            // if (fd == -1) {
            // perror("Open failed\n");
```

NORMAL ./signal_process_group.c unix utf-8 c 20% 56:59

assignment_8 : vi | assignment_8 : zsh | man : man

assignment_8 : vi — Konsole

File Edit View Bookmarks Settings Help

./signal_process_group.c

```c
        //  return errno;
        // }
        int num;
        long pgid_3;
        if ((num = read(fd, &pgid_3, sizeof(long))) == -1) {
            perror("Read failed\n");
        }
        else if (num < sizeof(long)) {
            printf("Expected number of bytes not read\n");
            exit(0);
        }
        printf("pgid_3 in process 5: %ld (to check FIFO transfer of pgid_3)\n", pgid_3);
        printf("(before setpgid with pgid_3) process 5 pgid: %ld\n", (long)getpgid(0));
        if (setpgid(0, (pid_t)pgid_3) == -1) {
            perror("Setpgid failed in process 5 for process 5\n");
            return errno;
        }
        printf("(after setpgid with pgid_3) process 5 pgid: %ld\n", (long)getpgid(0));
        /* checking the value of shared_pgid_3 to validate functioning of
         * mmap, ideally the value should be -1 as set by process 0
         * TODO: check if another possible value could be -2 in the scenario
         * when process 0 continues till the alarm expires and the sig_alrm
         * handler changes the value to -2 and process 5 executes
         * just before the if condition check following the break of while
         * loop in process 0
         */
        printf("Value pointed to by shared_pgid_3 (-1) in process 5 before it changes the value to pgid_3: %ld (to check working of mmap)\n", *shared_pgid_3);
        /* storing the value of pgid_3 in the shared memory region created
         * by process 0 pointed to by shared_pgid_3
         * this is being done here instead of doing it in process 3 to
         * ensure that process 5 has been shifted to the proces group
         * created by 3 beforehand so that it too receives SIGUSR1 signal
         * from process 0
         */
        *shared_pgid_3 = pgid_3;
        /* setting signal handler for SIGUSR1 in process 5
```

NORMAL ./signal_process_group.c unix utf-8 c 34% 96:60

assignment_8 : vi | assignment_8 : zsh | man : man

assignment_8 : vi — Konsole

File Edit View Bookmarks Settings Help

./signal_process_group.c

```c
         */
        if (signal(SIGUSR1, sig_usr1) == SIG_ERR) {
            perror("Signal failed for SIGUSR1 in process 5\n");
            return errno;
        }
        sleep(10);
        return 0;
    }
    else {
        printf("process 1 pid: %ld ppid: %ld\n", (long)getpid(), (long)getppid());
        sleep(10);
        return 0;
    }
}
else {
    if ((pid = fork()) == -1) {
        perror("Fork failed in process 0\n");
        return errno;
    }
    else if (pid == 0) {
        /* process 2 */
        if ((pid = fork()) == -1) {
            perror("Fork failed in process 2\n");
            return errno;
        }
        else if (pid == 0) {
            /* process 3 */
            if ((pid = fork()) == -1) {
                perror("Fork failed in process 3\n");
                return errno;
            }
            else if (pid == 0) {
                /* process 4 */
                printf("process 4 pid: %ld ppid: %ld\n", (long)getpid(), (long)getppid());
                /* if setpgid here gets called before the setpgid(0, 0) call
                 * in process 3, then pgid of process 4 will be set to the
```

NORMAL ./signal_process_group.c unix utf-8 c 43% 121:15

assignment_8 : vi | assignment_8 : zsh | man : man

assignment_8 : vi — Konsole

File Edit View Bookmarks Settings Help

./signal_process_group.c

```c
                 * old pgid of process 3, not its new one
                 */
                // setpgid(0, getpgid(getppid()));
                /* even if just printf is called here, if it gets called
                 * before the setpgid(pid, getpid(0)) call in process 3,
                 * then the pgid of process 4 will be printed as the old
                 * pgid instead of the new one
                 */
                // printf("process 4 pgid: %ld\n", (long)getpgid(0));
                /* setting signal handler for SIGUSR1 in process 4
                 */
                if (signal(SIGUSR1, sig_usr1) == SIG_ERR) {
                    perror("Signal failed for SIGUSR1 in process 4\n");
                    return errno;
                }
                sleep(10);
                return 0;
            }
            else {
                printf("process 3 pid: %ld ppid: %ld\n", (long)getpid(), (long)getppid());
                if (setpgid(0, 0) == -1) {
                    perror("Setpgid failed in process 3 for process 3\n");
                    return errno;
                }
                printf("process 3 pgid: %ld\n", (long)getpgid(0));
                if (setpgid(pid, getpgid(0)) == -1) {
                    perror("Setpgid failed in process 3 for process 4\n");
                }
                printf("process 4 pgid: %ld\n", (long)getpgid(pid));
                if (access(FIFO_NAME, F_OK) == -1) {
                    printf("In access\n");
                    if (mkfifo(FIFO_NAME, S_IFIFO | 0666) == -1) {
                        perror("Mkfifo failed\n");
                        return errno;
                    }
                }
```

NORMAL ./signal_process_group.c unix utf-8 c 62% 172:30

assignment_8 : vi | assignment_8 : zsh | man : man

assignment_8 : vi — Konsole

File Edit View Bookmarks Settings Help

./signal_process_group.c

```c
                int fd = creat(FIFO_NAME, 0666);
                if (fd == -1) {
                    perror("Creat failed\n");
                    return errno;
                }
                int num;
                long pgid_3 = (long)getpgid(0);
                printf("pgid_3 in process 3: %ld (to check FIFO transfer of pgid_3)\n", pgid_3);
                if ((num = write(fd, &pgid_3, sizeof(long))) == -1) {
                    perror("Write failed\n");
                    return errno;
                }
                else if (num < sizeof(long)) {
                    printf("Expected number of bytes not written\n");
                    exit(0);
                }
                if (close(fd) == -1) {
                    perror("Close failed in process 3\n");
                    return errno;
                }
                /* setting signal handler for SIGUSR1 in process 3
                 */
                if (signal(SIGUSR1, sig_usr1) == SIG_ERR) {
                    perror("Signal failed for SIGUSR1 in process 3\n");
                    return errno;
                }
                sleep(10);
                return 0;
            }
        }
        else {
            printf("process 2 pid: %ld ppid: %ld\n", (long)getpid(), (long)getppid());
            sleep(10);
            return 0;
        }
    }
```

NORMAL ./signal_process_group.c unix utf-8 c 76% 213:70

assignment_8 : vi | assignment_8 : zsh | man : man

```c
        else {
            printf("process 0 pid: %ld\n", (long)getpid());
            /* setting a signal handler for alarm and then setting an alarm to
             * break out of the while loop by changing the value of
             * shared_pgid_3 to -2 in the signal handler for alarm
             */
            if (signal(SIGALRM, sig_alrm) == SIG_ERR) {
                perror("Signal failed for SIGALRM in process 0\n");
                return errno;
            }
            alarm(5);
            while (*shared_pgid_3 == -1);
            alarm(0);
            if (*shared_pgid_3 == -2) {
                printf("Stuck in while for too long, shared memory transfer failed, signal to process group created by 3 not generated\n");
                return 0;
            }
            printf("Value pointed to by shared_pgid_3 (pgid_3) in process 0: %ld (to check working of mmap)\n", *shared_pgid_3);
            printf("Sending SIGUSR1 signal to the process group created by 3...\n");
            /* sending the SIGUSR1 signal to process group created by 3
             * kill succeeds here i.e. in process 0 since its real uid equals
             * those of all processes in the process group created by 3 i.e. 3,
             * 4 and 5
             */
            if (kill((pid_t)(-(*shared_pgid_3)), SIGUSR1) == -1) {
                perror("Kill failed\n");
                return errno;
            }
            else {
                printf("SIGUSR1 signal sent to the process group created by 3 successfully\n");
            }
            /* to undo the shared memory setting using shared_pgid_3 made by mmap
             * though this region will automatically get unmapped after process
             * 0 terminates
             */
            if (munmap((void *)shared_pgid_3, (size_t)sizeof(long)) == -1) {
```

```c
                perror("Munmap failed\n");
                return errno;
            }
            if (shared_pgid_3 == NULL) {
                printf("Shared memory region reference pointer shared_pgid_3 in process 0: NULL (to check working of munmap)\n");
            }
            /* if uncommented, this causes a segmentation fault to occur due to
             * invalid memory reference as a result of calling munmap
             */
            printf("Value pointed to by shared_pgid_3 in process 0: %ld (to check working of munmap)\n", *shared_pgid_3);
            sleep(10);
        }
    }
    return 0;
}
```

Output-

```
niramay@niramay  ~/Documents/sem7/AUP/assignment_8  ./signal_process_group
process 0 pid: 7038
process 2 pid: 7040 ppid: 7038
process 1 pid: 7039 ppid: 7038
process 5 pid: 7041 ppid: 7039
process 3 pid: 7042 ppid: 7040
process 4 pid: 7043 ppid: 7042
process 3 pgid: 7042
process 4 pgid: 7042
pgid_3 in process 3: 7042 (to check FIFO transfer of pgid_3)
pgid_3 in process 5: 7042 (to check FIFO transfer of pgid_3)
(before setpgid with pgid_3) process 5 pgid: 7038
(after setpgid with pgid_3) process 5 pgid: 7042
Value pointed to by shared_pgid_3 (-1) in process 5 before it changes the value to pgid_3: -1 (to check working of mmap)
Value pointed to by shared_pgid_3 (pgid_3) in process 0: 7042 (to check working of mmap)
Sending SIGUSR1 signal to the process group created by 3...
SIGUSR1 signal sent to the process group created by 3 successfully
SIGUSR1 (10) received, pid = 7041
SIGUSR1 (10) received, pid = 7042
SIGUSR1 (10) received, pid = 7043
niramay@niramay  ~/Documents/sem7/AUP/assignment_8
```

```
niramay@niramay  ~/Documents/sem7/AUP/assignment_8  ./signal_process_group
process 0 pid: 7235
process 1 pid: 7236 ppid: 7235
process 2 pid: 7237 ppid: 7235
process 5 pid: 7239 ppid: 7236
process 4 pid: 7240 ppid: 7238
process 3 pid: 7238 ppid: 7237
process 3 pgid: 7238
process 4 pgid: 7238
pgid_3 in process 3: 7238 (to check FIFO transfer of pgid_3)
pgid_3 in process 5: 7238 (to check FIFO transfer of pgid_3)
(before setpgid with pgid_3) process 5 pgid: 7235
(after setpgid with pgid_3) process 5 pgid: 7238
Value pointed to by shared_pgid_3 (-1) in process 5 before it changes the value to pgid_3: -1 (to check working of mmap)
Value pointed to by shared_pgid_3 (pgid_3) in process 0: 7238 (to check working of mmap)
Sending SIGUSR1 signal to the process group created by 3...
SIGUSR1 signal sent to the process group created by 3 successfully
SIGUSR1 (10) received, pid = 7240
SIGUSR1 (10) received, pid = 7239
SIGUSR1 (10) received, pid = 7238
[1]    7235 segmentation fault (core dumped)  ./signal_process_group
niramay@niramay  ~/Documents/sem7/AUP/assignment_8  cc -g signal_process_group.c -o signal_process_group
```

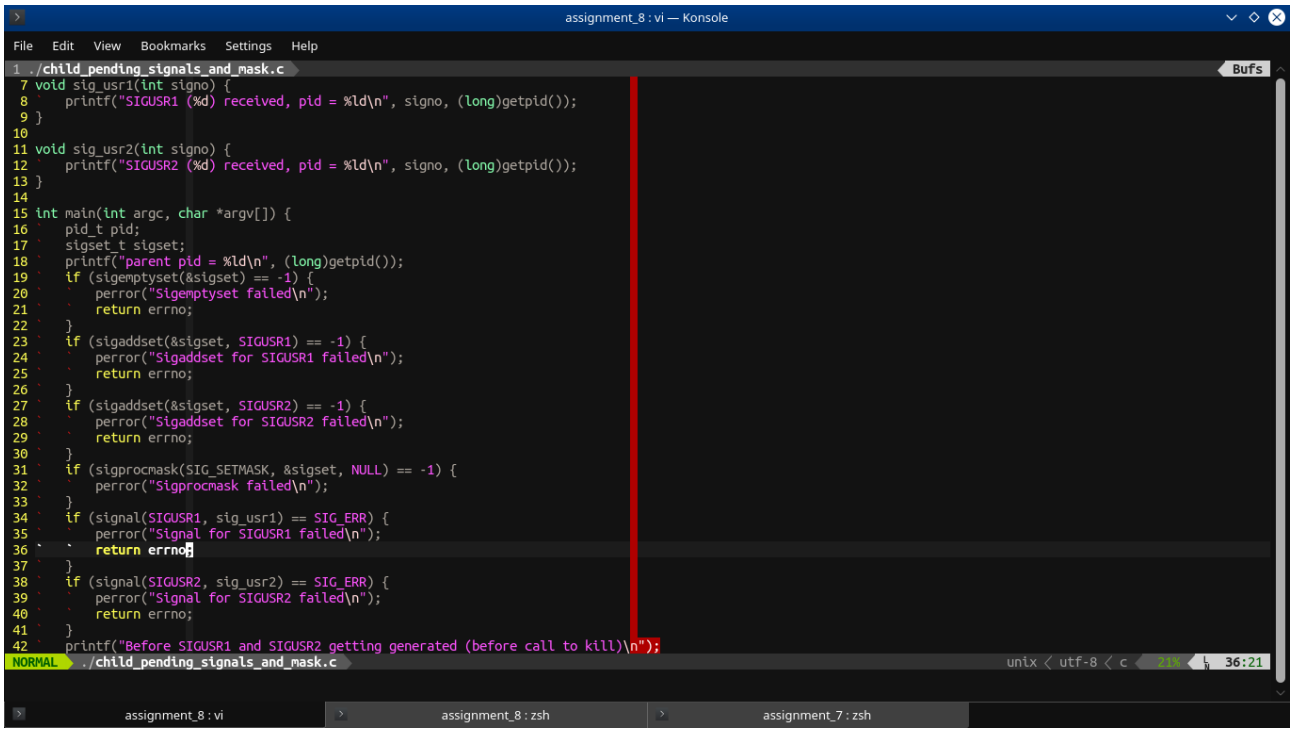
Explanation-

Mmap was used to create a shared memory region in process 0 for it to receive the process group id of the process group created by 3 (which contains processes 3, 4 and 5 as per coded in the previous assignment) as opposed to using a FIFO to transfer the pgid. Because of this, the obtained pointer to this shared memory region was copied down the process tree and was made available in process 5 which stored the pgid of the said group at this location. This pgid value was then used by process 0 to generate a SIGUSR1 signal to this process group. The output shows SIGUSR1 received three times because there are 3 processes in this process group and all of them received this signal. The rest of the print statements are merely for understanding code flow and debugging purposes. Correct FIFO transfer of pgid from process 3 to process 5 is checked, pids and ppids of all processes created are outputted, processes whose process group ids change are outputted again to display the changed pgid values, correct working of mmap is checked and finally the segmentation fault in the second output image indicates correct functioning of munmap which undoes the creation of the shared memory region and deems all future references to it as invalid memory accesses.

TODO- in process 5, check if another possible value of shared_pgid_3 could be -2 in the scenario when process 0 continues till the alarm expires and the sig_alrm handler changes the value to -2 and process 5 executes just before the if condition check following the break of while loop in process 0 (commented in code as well) (the outptut screenshots do not show the occurence of this case as seen in the first print statement saying 'to check working of mmap', -1 is observed, not -2).

3. “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-



assignment_8 : vi — Konsole

File Edit View Bookmarks Settings Help

./child_pending_signals_and_mask.c

```c
    if (sigismember(&sigset, SIGUSR1)) {
        printf("SIGUSR1 has been blocked in parent\n");
    }
    if (sigismember(&sigset, SIGUSR2)) {
        printf("SIGUSR2 has been blocked in parent\n");
    }
    /* this will generate the SIGUSR1 and SIGUSR2 signals in parent but since
     * they are blocked they will remain in the pending state
     */
    if (kill(getpid(), SIGUSR1) == -1) {
        perror("Kill for SIGUSR1 failed\n");
        return errno;
    }
    if (kill(getpid(), SIGUSR2) == -1) {
        perror("Kill for SIGUSR2 failed\n");
        return errno;
    }
    sigset_t sigset_pending;
    if (sigpending(&sigset_pending) == -1) {
        perror("Sigpending failed\n");
        return errno;
    }
    printf("After SIGUSR1 and SIGUSR2 getting generated (after call to kill)\n");
    if (sigismember(&sigset_pending, SIGUSR1)) {
        printf("SIGUSR1 is pending in parent\n");
    }
    else {
        printf("SIGUSR1 is not pending in parent\n");
    }
    if (sigismember(&sigset_pending, SIGUSR2)) {
        printf("SIGUSR2 is pending in parent\n");
    }
    else {
        printf("SIGUSR2 is not pending in parent\n");
    }
    /* this will remove SIGUSR1 from being blocked and hence the signal will get
```

NORMAL ./child_pending_signals_and_mask.c unix utf-8 c 35% 61:44

assignment_8 : vi | assignment_8 : zsh | assignment_7 : zsh

assignment_8 : vi — Konsole

File Edit View Bookmarks Settings Help

./child_pending_signals_and_mask.c

```c
     * out of its pending state to be delivered to the parent once sigprocmask
     * is called to set this new mask without SIGUSR1 blocked
     */
    if (sigdelset(&sigset, SIGUSR1) == -1) {
        perror("Sigdelset failed\n");
        return errno;
    }
    if (sigprocmask(SIG_SETMASK, &sigset, NULL) == -1) {
        perror("Sigprocmask failed\n");
    }
    if (sigpending(&sigset_pending) == -1) {
        perror("Sigpending failed\n");
        return errno;
    }
    printf("After SIGUSR1 is removed from being blocked but SIGUSR2 is not (after call to sigdelset)\n");
    if (sigismember(&sigset_pending, SIGUSR1)) {
        printf("SIGUSR1 is pending in parent\n");
    }
    else {
        printf("SIGUSR1 is not pending in parent\n");
    }
    if (sigismember(&sigset_pending, SIGUSR2)) {
        printf("SIGUSR2 is pending in parent\n");
    }
    else {
        printf("SIGUSR2 is not pending in parent\n");
    }
    if ((pid = fork()) == -1) {
        perror("Fork failed\n");
        return errno;
    }
    else if (pid == 0) {
        printf("child pid = %ld\n", (long )getpid());
        /* the child's signal mask will be inherited from the parent and will
         * contain SIGUSR2 but not SIGUSR1, hence SIGUSR2 will be blocked but
         * not SIGUSR1
```

NORMAL ./child_pending_signals_and_mask.c unix utf-8 c 45% 79:78

assignment_8 : vi | assignment_8 : zsh | assignment_7 : zsh

```c
     */
    if (sigismember(&sigset, SIGUSR1)) {
        printf("SIGUSR1 has been blocked in child\n");
    }
    else {
        printf("SIGUSR1 has not been blocked in child\n");
    }
    if (sigismember(&sigset, SIGUSR2)) {
        printf("SIGUSR2 has been blocked in child\n");
    }
    else {
        printf("SIGUSR2 has not been blocked in child\n");
    }
    /* since pending signals from the parent are not inherited by the child,
     * if SIGUSR2 is unblocked in the child, it will not be delivered to the
     * child
     */
    if (sigpending(&sigset_pending) == -1) {
        perror("Sigpending failed\n");
        return errno;
    }
    if (sigismember(&sigset_pending, SIGUSR1)) {
        printf("SIGUSR1 is pending in child\n");
    }
    else {
        printf("SIGUSR1 is not pending in child\n");
    }
    if (sigismember(&sigset_pending, SIGUSR2)) {
        printf("SIGUSR2 is pending in child\n");
    }
    else {
        printf("SIGUSR2 is not pending in child\n");
    }
    if (sigdelset(&sigset, SIGUSR2) == -1) {
        perror("Sigdelset failed\n");
        return errno;
```

```c
    }
    if (sigprocmask(SIG_SETMASK, &sigset, NULL) == -1) {
        perror("Sigprocmask failed\n");
    }
    if (sigpending(&sigset_pending) == -1) {
        perror("Sigpending failed\n");
        return errno;
    }
    printf("After SIGUSR1 is removed from being blocked (after call to sigdelset)\n");
    if (sigismember(&sigset_pending, SIGUSR1)) {
        printf("SIGUSR1 is pending in child\n");
    }
    else {
        printf("SIGUSR1 is not pending in child\n");
    }
    if (sigismember(&sigset_pending, SIGUSR2)) {
        printf("SIGUSR2 is pending in child\n");
    }
    else {
        printf("SIGUSR2 is not pending in child\n");
    }
  }
  return 0;
}
```

Output-

```
nitramay@nitramay ~/Documents/sem7/AUP/assignment_8 ./child_pending_signals_and_mask
parent pid = 11879
Before SIGUSR1 and SIGUSR2 getting generated (before call to kill)
SIGUSR1 has been blocked in parent
SIGUSR2 has been blocked in parent
After SIGUSR1 and SIGUSR2 getting generated (after call to kill)
SIGUSR1 is pending in parent
SIGUSR2 is pending in parent
SIGUSR1 (10) received, pid = 11879
After SIGUSR1 is removed from being blocked but SIGUSR2 is not (after call to sigdelset)
SIGUSR1 is not pending in parent
SIGUSR2 is pending in parent
child pid = 11880
SIGUSR1 has not been blocked in child
SIGUSR2 has been blocked in child
SIGUSR1 is not pending in child
SIGUSR2 is not pending in child
After SIGUSR1 is removed from being blocked (after call to sigdelset)
SIGUSR1 is not pending in child
SIGUSR2 is not pending in child
nitramay@nitramay ~/Documents/sem7/AUP/assignment_8
```

Explanation-

SIGUSR1 and SIGUSR2 were blocked in the parent. Then they were generated which put them in the pending state. Then SIGUSR1 was removed from being blocked. As a result, it was delivered to the parent. Hence, in the child, the signal mask was such that SIGUSR1 was not blocked but SIGUSR2 was blocked since this signal mask was inherited from the parent. Though, both of them

were not in the pending state since pending signals are not inherited from the parent (SIGUSR2 was pending in the parent). Hence, even when SIGUSR2 was removed from being blocked , it was not delivered to the child and sig_usr2 i.e. its signal handler wasn't called.