ASSIGNMENT 3
29/08/23

NAME : SHRESTH SONKAR
REGNO : 20214272
GROUP : CS5D
TOPIC : OS LAB
CODE : CS-15203

```c
/* Q1
 * Write a program in C that creates a child process,
 * waits for the termination of the child and lists
process ID (PID),
 * together with the state in which the process was
terminated (in decimal and hexadecimal).
 */


#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>

int main() {
    pid_t child_pid;
    int status;
    child_pid = fork();

    if (child_pid < 0)
        perror("Fork failed!\n");
    else if (child_pid == 0) {
        printf("Child process (PID : %d) is running\n",
getpid());
        exit(42);
    } else {
        waitpid(child_pid, &status, 0);
        printf("Child process (PID : %d) has
terminated\n", child_pid);
        if (WIFEXITED(status)) {
            printf("Termination status (DEC) : %d\n",
WIFEXITED(status));
            printf("Termination status (HEX) : 0x%X\n",
WIFEXITED(status));
        } else {
            printf("Child process did not terminate
normally!\n");
        }
    }
    return 0;
}
```

```c
/* Q2
 * Write a program where child process sleeps for
'2'seconds while the parent process waits for the child
process to exit.
 * Note how return value of fork is used to control
which code is run by parent and which by the child.
 * (Use if required:fork (), exec( ), wait( ),
signal( ), kill( ), alarm( )system calls).
 */


#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>

int main() {
    pid_t child_pid;
    int status;
    child_pid = fork();

    if (child_pid < 0) {
        perror("Fork failed!\n");
        exit(1);
    } else if (child_pid == 0) {
        printf("Child process (PID : %d) is sleeping...
\n", getpid());
        sleep(2);
        printf("Child process (PID : %d) is running\n",
getpid());
        exit(0);
    } else {
        printf("Parent process (PID : %d) is waiting
for child process to exit.\n", getpid());
        waitpid(child_pid, NULL, 0);
        printf("Parent process (PID : %d) child has
exited.\n", getpid());
    }
    return 0;
}
```

```c
/* Q3
 * Write a program that calls fork()system call.
 * Before calling fork(), have the main process access
a variable (e.g.,x) and set its value to something
(e.g.,1000).
 * What value is the variable in the child process?
 * What happens to the variable when both the child and
parent change the value of x?
 * Clearly write your observations.
 */


#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>

int main() {
    int x = 1000;
    pid_t child_pid = fork();

    if (child_pid < 0) {
        perror("Fork failed!\n");
        exit(1);
    } else if (child_pid == 0) {
        printf("Child process before modification: x =
%d\n", x);
        x = 2000;
        printf("Child process after modification: x =
%d\n", x);
    } else {
        printf("Parent process before modification: x =
%d\n", x);
        x = 3000;
        printf("Parent process after modification: x =
%d\n", x);
    }
    return 0;
}
```

```c
/* Q4
 * Write a program that opens a file (with the open()
system call)
 * and then calls fork()to create a new process.
 * Can both the child and parent access the file
descriptor returned by open()?
 * What happens when they are writing to the file
concurrently, i.e., at the same time?
 * Clearly write your observations.
 */

#include <stdio.h>
#include <fcntl.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>

int main() {
    int fd;
    pid_t child_pid;
    fd = open("output.txt", O_WRONLY | O_CREAT |
O_TRUNC, 0644);

    if (fd == -1) {
        perror("open");
        exit(EXIT_FAILURE);
    }
    child_pid = fork();

    if (child_pid < 0) {
        perror("Fork failed!\n");
        exit(1);
    } else if (child_pid == 0) {
        const char *child_message = "Hello from child!
\n";
        write(fd, child_message,
strlen(child_message));
    } else {
        const char *parent_message = "Hello from
parent!\n";
        write(fd, parent_message,
strlen(parent_message));
```

```
    }

    close(fd);
    system("cat output.txt");
    return 0;
}
```

```c
/* Q5
 * In a C program, print the address of the variable
and enter into a long loop (say using while (1)).
 * a.Start three to four processes of the same program
and observe the printed address values.
 * b.Show how two processes which are members of the
relationship parent-child are concurrent from execution
point of view,
 * initially the child is copy of the parent, but every
process has its own data.
 */


#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main() {
    int x = 0;
    int data = 42;
    printf("Parent Process ID: %d\n", getpid());
    printf("Parent - Address of data: %p\n", &data);

    for (int i = 0; i < 3; i++) {
        pid_t child_pid = fork();

        if (child_pid < 0) {
            perror("Fork failed");
            exit(1);
        } else if (child_pid == 0) {
            printf("Child Process ID: %d\n", getpid());
            printf("Child - Address of data: %p\n",
&data);
            data = i * 10;

            while (x < 1000000000) {
                x++;
            }

            printf("Child %d - Completed\n", i);
            exit(0);
        }
    }
```

```
    while (x < 1000000000) {
        x++;
    }

    printf("Parent - Completed\n");
    system("sleep 1");
    return 0;
}
```

```c
/* Q6
 * Write a program that creates a child process.
 * Parent process writes data to 'pipe'
 * child process reads the data from pipe
 * prints data on the screen.
 */

#include <stdio.h>
#include <fcntl.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>

int main() {
    int pipe_fd[2];
    char buff[100];

    if (pipe(pipe_fd) == -1) {
        perror("Pipe creation failed!\n");
        exit(1);
    }
    pid_t child_pid = fork();

    if (child_pid < 0) {
        perror("Fork failed!\n");
        exit(1);
    } else if (child_pid == 0) {
        close(pipe_fd[1]);
        ssize_t bytes_read = read(pipe_fd[0], buff,
sizeof(buff));

        if (bytes_read > 0) {
            printf("Child process: \n\tReceived data
from parent: %.*s\n", (int) bytes_read, buff);
        } else {
            printf("Child process: \n\tFailed to
receive data from parent\n");
        }
        close(pipe_fd[0]);
    } else {
        close(pipe_fd[0]);
```

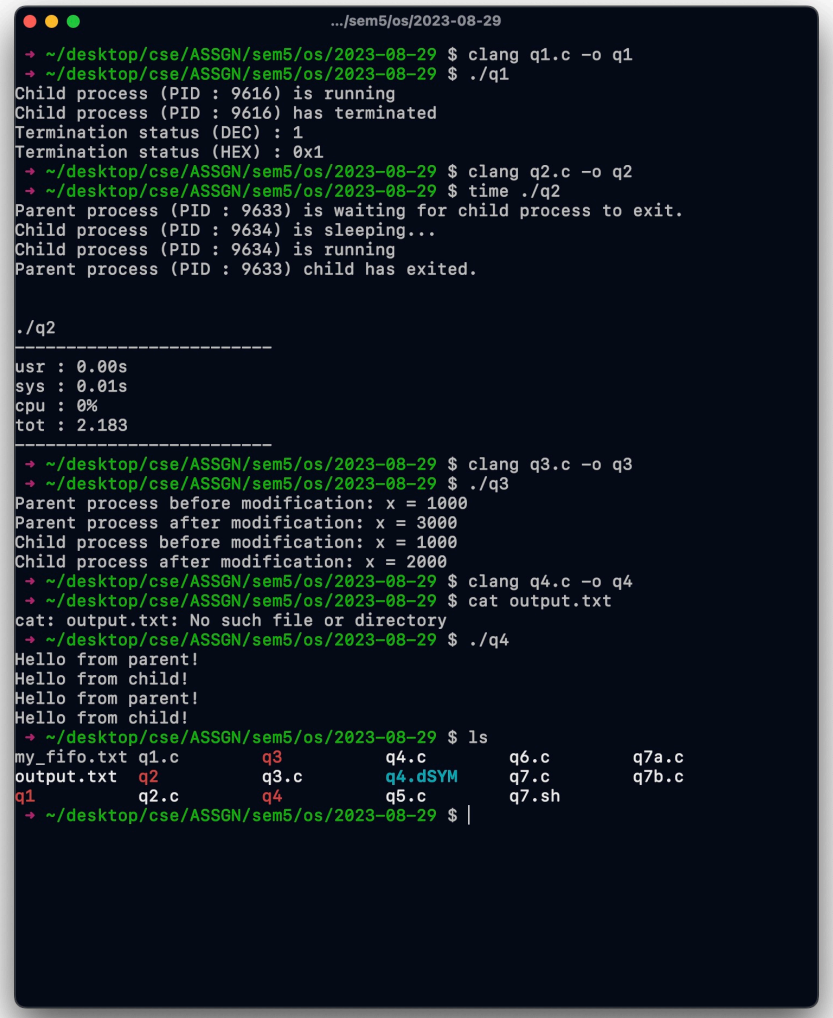
```c
        const char *dataToSend = "Hello from Parent";
        ssize_t bytes_written = write(pipe_fd[1],
dataToSend, strlen(dataToSend));

        if (bytes_written > 0) {
            printf("Parent process: \n\tSent data to
Child: %s\n", dataToSend);
        } else {
            printf("Parent process: \n\tFailed to send
data to Child\n");
        }
        close(pipe_fd[1]);
        wait(NULL);
    }
    return 0;
}
```

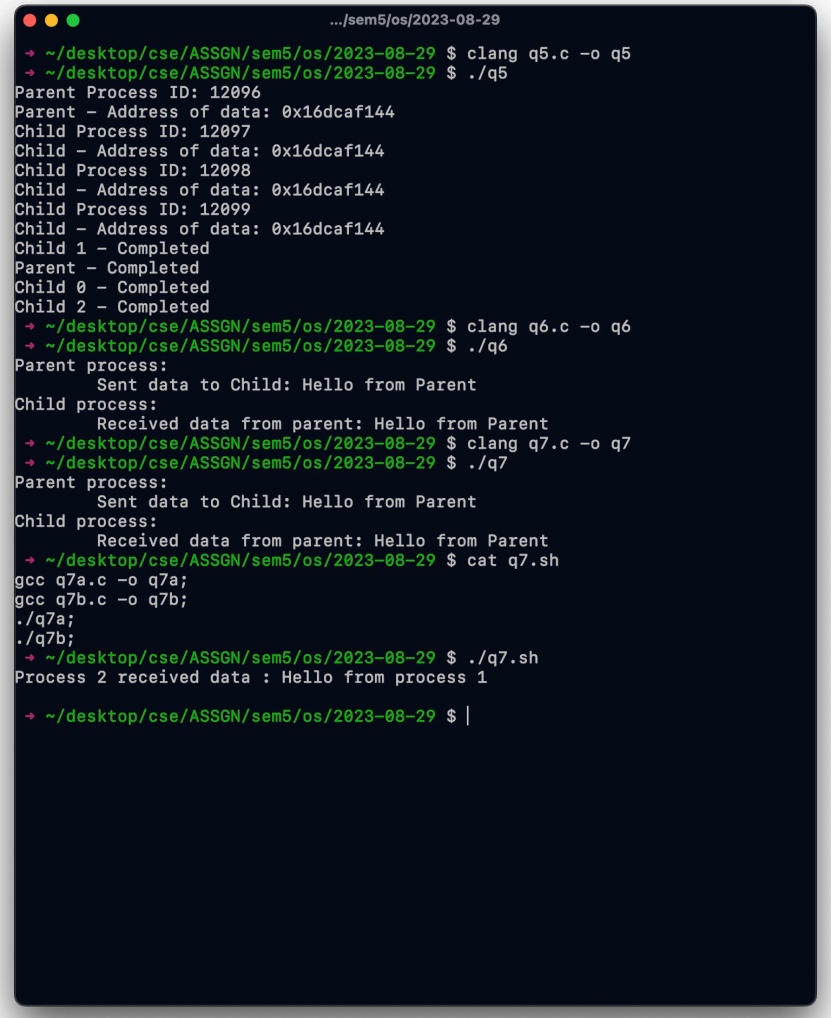
```c
/* Q7a
 * Implement inter process communication when two
processes are related
 */

#include <stdio.h>
#include <fcntl.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>

int main() {
    int pipe_fd[2];
    char buff[100];

    if (pipe(pipe_fd) == -1) {
        perror("Pipe creation failed!\n");
        exit(1);
    }
    pid_t child_pid = fork();

    if (child_pid < 0) {
        perror("Fork failed!\n");
        exit(1);
    } else if (child_pid == 0) {
        close(pipe_fd[1]);
        ssize_t bytes_read = read(pipe_fd[0], buff,
sizeof(buff));

        if (bytes_read > 0) {
            printf("Child process: \n\tReceived data
from parent: %.*s\n", (int) bytes_read, buff);
        } else {
            printf("Child process: \n\tFailed to
receive data from parent\n");
        }
        close(pipe_fd[0]);
    } else {
        close(pipe_fd[0]);
        const char *dataToSend = "Hello from Parent";
        ssize_t bytes_written = write(pipe_fd[1],
dataToSend, strlen(dataToSend));
```

```
        if (bytes_written > 0) {
            printf("Parent process: \n\tSent data to
Child: %s\n", dataToSend);
        } else {
            printf("Parent process: \n\tFailed to send
data to Child\n");
        }
        close(pipe_fd[1]);
        wait(NULL);
    }
    return 0;
}
```

```c
/* Q7b
 * Implement inter process communication when two
processes are not related
 */

#include <stdio.h>
#include <fcntl.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>

int main() {
    int fd;
    fd = open("my_fifo.txt", O_WRONLY);

    if (fd == -1) {
        perror("Failed to open FIFO for writing!\n");
        exit(1);
    }

    const char *dataToSend = "Hello from process 1\n";
    write(fd, dataToSend, strlen(dataToSend) + 1);
    close(fd);
    return 0;
}
```

```c
#include <stdio.h>
#include <fcntl.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/types.h>

int main() {
    int fd;
    char buff[100];
    fd = open("my_fifo.txt", O_RDONLY);

    if (fd == -1) {
        perror("Failed to open FIFO for reading!\n");
        exit(1);
    }

    ssize_t bytes_read = read(fd, buff, sizeof(buff));
    if (bytes_read > 0)
        printf("Process 2 received data : %s\n", buff);

    close(fd);
    return 0;
}
```


.../sem5/os/2023-08-29
→ ~/desktop/cse/ASSGN/sem5/os/2023-08-29 $ clang q1.c -o q1
→ ~/desktop/cse/ASSGN/sem5/os/2023-08-29 $ ./q1
Child process (PID : 9616) is running
Child process (PID : 9616) has terminated
Termination status (DEC) : 1
Termination status (HEX) : 0x1
→ ~/desktop/cse/ASSGN/sem5/os/2023-08-29 $ clang q2.c -o q2
→ ~/desktop/cse/ASSGN/sem5/os/2023-08-29 $ time ./q2
Parent process (PID : 9633) is waiting for child process to exit.
Child process (PID : 9634) is sleeping...
Child process (PID : 9634) is running
Parent process (PID : 9633) child has exited.
./q2
usr : 0.00s
sys : 0.01s
cpu : 0%
tot : 2.183
→ ~/desktop/cse/ASSGN/sem5/os/2023-08-29 $ clang q3.c -o q3
→ ~/desktop/cse/ASSGN/sem5/os/2023-08-29 $ ./q3
Parent process before modification: x = 1000
Parent process after modification: x = 3000
Child process before modification: x = 1000
Child process after modification: x = 2000
→ ~/desktop/cse/ASSGN/sem5/os/2023-08-29 $ clang q4.c -o q4
→ ~/desktop/cse/ASSGN/sem5/os/2023-08-29 $ cat output.txt
cat: output.txt: No such file or directory
→ ~/desktop/cse/ASSGN/sem5/os/2023-08-29 $ ./q4
Hello from parent!
Hello from child!
Hello from parent!
Hello from child!
→ ~/desktop/cse/ASSGN/sem5/os/2023-08-29 $ ls
my_fifo.txt q1.c q3 q4.c q6.c q7a.c
output.txt q2 q3.c q4.dSYM q7.c q7b.c
q1 q2.c q4 q5.c q7.sh
→ ~/desktop/cse/ASSGN/sem5/os/2023-08-29 $


.../sem5/os/2023-08-29
→ ~/desktop/cse/ASSGN/sem5/os/2023-08-29 $ clang q5.c -o q5
→ ~/desktop/cse/ASSGN/sem5/os/2023-08-29 $ ./q5
Parent Process ID: 12096
Parent - Address of data: 0x16dcaf144
Child Process ID: 12097
Child - Address of data: 0x16dcaf144
Child Process ID: 12098
Child - Address of data: 0x16dcaf144
Child Process ID: 12099
Child - Address of data: 0x16dcaf144
Child 1 - Completed
Parent - Completed
Child 0 - Completed
Child 2 - Completed
→ ~/desktop/cse/ASSGN/sem5/os/2023-08-29 $ clang q6.c -o q6
→ ~/desktop/cse/ASSGN/sem5/os/2023-08-29 $ ./q6
Parent process:
Sent data to Child: Hello from Parent
Child process:
Received data from parent: Hello from Parent
→ ~/desktop/cse/ASSGN/sem5/os/2023-08-29 $ clang q7.c -o q7
→ ~/desktop/cse/ASSGN/sem5/os/2023-08-29 $ ./q7
Parent process:
Sent data to Child: Hello from Parent
Child process:
Received data from parent: Hello from Parent
→ ~/desktop/cse/ASSGN/sem5/os/2023-08-29 $ cat q7.sh
gcc q7a.c -o q7a;
gcc q7b.c -o q7b;
./q7a;
./q7b;
→ ~/desktop/cse/ASSGN/sem5/os/2023-08-29 $ ./q7.sh
Process 2 received data : Hello from process 1
→ ~/desktop/cse/ASSGN/sem5/os/2023-08-29 $