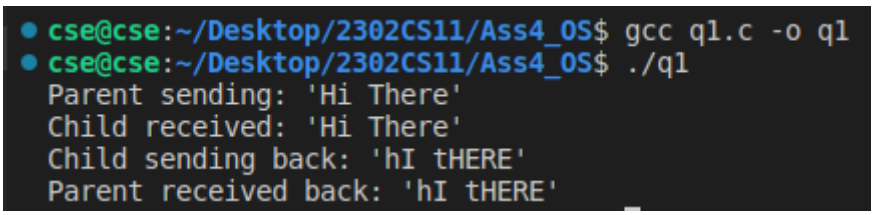
Indian Institute of Technology, Patna
Department of Computer Science and Engineering
CS3101: Operating System
Lab Assignment 4: Process Chains, Multi-Threading, and Inter-Proces Communication

Name: Aditya Prakash
Roll No. 2302CS11
Cubicle No. 150
TA Name : Kijen

q1.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <ctype.h>
#include <sys/wait.h>

int main() {
   int parent_to_child_fd[2];
   int child_to_parent_fd[2];

   if (pipe(parent_to_child_fd) == -1) {
      perror("pipe");
      exit(EXIT_FAILURE);
   }
   if (pipe(child_to_parent_fd) == -1) {
      perror("pipe");
      exit(EXIT_FAILURE);
   }

   pid_t pid = fork();

   if (pid < 0) {
      perror("fork");
      exit(EXIT_FAILURE);
   }

   if (pid == 0) {
      close(parent_to_child_fd[1]);
      close(child_to_parent_fd[0]);

      char received_msg[100];
      read(parent_to_child_fd[0], received_msg, sizeof(received_msg));
      printf("Child received: '%s'\n", received_msg);

      for (int i = 0; i < strlen(received_msg); i++) {
         if (isupper(received_msg[i])) {
            received_msg[i] = tolower(received_msg[i]);
         } else if (islower(received_msg[i])) {
            received_msg[i] = toupper(received_msg[i]);
```

```
        }
    }

    printf("Child sending back: '%s'\n", received_msg);
    write(child_to_parent_fd[1], received_msg, strlen(received_msg) + 1);

    close(parent_to_child_fd[0]);
    close(child_to_parent_fd[1]);

    exit(EXIT_SUCCESS);

} else {
    close(parent_to_child_fd[0]);
    close(child_to_parent_fd[1]);
    char* msg_to_send = "Hi There";
    printf("Parent sending: '%s'\n", msg_to_send);
    write(parent_to_child_fd[1], msg_to_send, strlen(msg_to_send) + 1);

    wait(NULL);

    char received_back_msg[100];
    read(child_to_parent_fd[0], received_back_msg, sizeof(received_back_msg));
    printf("Parent received back: '%s'\n", received_back_msg);

    close(parent_to_child_fd[1]);
    close(child_to_parent_fd[0]);
}

return 0;
}
```



q2a.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <ctype.h>
#include <sys/wait.h>

int main(int argc, char *argv[]) {
    int n_to_send;
    if(argc>1){
        n_to_send = atoi(argv[1]);
```

```
}
else{
    printf("Argument was not passed");
    return -1;
}
int parent_to_child_fd[2];

if (pipe(parent_to_child_fd) == -1) {
    perror("pipe");
    exit(EXIT_FAILURE);
}

pid_t pid = fork();

if (pid < 0) {
    perror("fork");
    exit(EXIT_FAILURE);
}

if (pid == 0) {
    close(parent_to_child_fd[1]);

    int n_terms;
    read(parent_to_child_fd[0], &n_terms, sizeof(n_terms));
    printf("Child received terms count: %d\n", n_terms);
    printf("Child generating Fibonacci sequence:\n");

    if (n_terms > 0) {
        long long first = 0, second = 1;
        if (n_terms >= 1) {
            printf("%lld ", first);
        }
        if (n_terms >= 2) {
            printf("%lld ", second);
        }
        for (int i = 2; i < n_terms; i++) {
            long long next = first + second;
            printf("%lld ", next);
            first = second;
            second = next;
        }
    }

    printf("\nChild process completed.\n");

    close(parent_to_child_fd[0]);

    exit(EXIT_SUCCESS);

} else {
    close(parent_to_child_fd[0]);
```

```
        write(parent_to_child_fd[1], &n_to_send, sizeof(n_to_send));
        printf("Parent sent %d to child.\n", n_to_send);
        close(parent_to_child_fd[1]);

        wait(NULL);
        printf("Parent process finished.\n");
    }

    return 0;
}
```

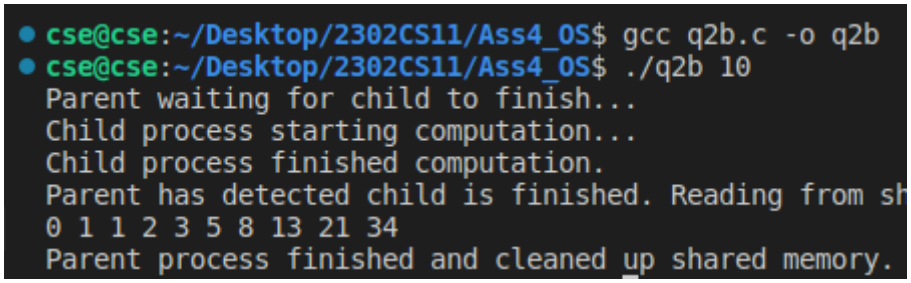
```
cse@cse:~/Desktop/2302CS11/Ass4_OS$ gcc q2a.c -o q2a
cse@cse:~/Desktop/2302CS11/Ass4_OS$ ./q2a 10
Parent sent 10 to child.
Child received terms count: 10
Child generating Fibonacci sequence:
0 1 1 2 3 5 8 13 21 34
Child process completed.
Parent process finished.
```

q2b.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/wait.h>
#include <sys/stat.h>

#define MAX_SEQUENCE 100

typedef struct {
    long long fib_sequence[MAX_SEQUENCE];
    int sequence_size;
} shared_data;

int main(int argc, char *argv[]) {
    int n_terms;
    if(argc>1){
        n_terms = atoi(argv[1]);
    }
    else{
        printf("Int agr was not properly passed!");
        return -1;
    }

    const char* shm_name = "/fibonacci_shm";
    int shm_fd;
    shared_data* shm_ptr;

    shm_fd = shm_open(shm_name, O_CREAT | O_RDWR, 0666);
```

```
    if (shm_fd == -1) {
        perror("shm_open");
        exit(EXIT_FAILURE);
    }

    ftruncate(shm_fd, sizeof(shared_data));

    shm_ptr = mmap(0, sizeof(shared_data), PROT_READ | PROT_WRITE, MAP_SHARED,
shm_fd, 0);
    if (shm_ptr == MAP_FAILED) {
        perror("mmap");
        exit(EXIT_FAILURE);
    }

    if (n_terms > MAX_SEQUENCE) {
        printf("Number exceeds maximum. Setting to %d.\n", MAX_SEQUENCE);
        n_terms = MAX_SEQUENCE;
    }
    if (n_terms < 0) {
        printf("Cannot generate a negative number of terms. Setting to 0.\n");
        n_terms = 0;
    }

    shm_ptr->sequence_size = n_terms;

    pid_t pid = fork();

    if (pid < 0) {
        perror("fork");
        exit(EXIT_FAILURE);
    }

    if (pid == 0) {

        printf("Child process starting computation...\n");

        if (shm_ptr->sequence_size > 0) {
            long long first = 0, second = 1;
            if (shm_ptr->sequence_size >= 1) {
                shm_ptr->fib_sequence[0] = first;
            }
            if (shm_ptr->sequence_size >= 2) {
                shm_ptr->fib_sequence[1] = second;
            }
            for (int i = 2; i < shm_ptr->sequence_size; i++) {
                long long next = first + second;
                shm_ptr->fib_sequence[i] = next;
                first = second;
                second = next;
            }
        }
```

```
        printf("Child process finished computation.\n");

        munmap(shm_ptr, sizeof(shared_data));

        exit(EXIT_SUCCESS);

    } else {

        printf("Parent waiting for child to finish...\n");
        wait(NULL);
        printf("Parent has detected child is finished. Reading from shared memory:\n");

        for (int i = 0; i < shm_ptr->sequence_size; i++) {
            printf("%lld ", shm_ptr->fib_sequence[i]);
        }
        printf("\n");

        if (munmap(shm_ptr, sizeof(shared_data)) == -1) {
            perror("munmap");
        }

        if (close(shm_fd) == -1) {
            perror("close");
        }

        if (shm_unlink(shm_name) == -1) {
            perror("shm_unlink");
        }

        printf("Parent process finished and cleaned up shared memory.\n");
    }

    return 0;
}
```



editor.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/select.h>
#include <signal.h>
```

```
#define MAX_JOURNALISTS 10
#define ARTICLE_BUFFER_SIZE 256

int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <num_journalists> <num_articles>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    int num_journalists = atoi(argv[1]);
    int num_articles_target = atoi(argv[2]);

    if (num_journalists <= 0 || num_journalists > MAX_JOURNALISTS) {
        fprintf(stderr, "Number of journalists must be between 1 and %d.\n", MAX_JOURNALISTS);
        exit(EXIT_FAILURE);
    }
    if (num_articles_target <= 0) {
        fprintf(stderr, "Number of articles must be positive.\n");
        exit(EXIT_FAILURE);
    }

    printf("Editor: Newsroom open. Waiting for %d articles from %d journalists.\n",
num_articles_target, num_journalists);
    fflush(stdout);

    int pipes[MAX_JOURNALISTS][2];
    pid_t pids[MAX_JOURNALISTS];

    for (int i = 0; i < num_journalists; i++) {
        if (pipe(pipes[i]) == -1) {
            perror("pipe");
            exit(EXIT_FAILURE);
        }

        pids[i] = fork();
        if (pids[i] == -1) {
            perror("fork");
            exit(EXIT_FAILURE);
        }

        if (pids[i] == 0) {
            close(pipes[i][0]);

            char journalist_id_str[10];
            char pipe_fd_str[10];
            sprintf(journalist_id_str, "%d", i + 1);
            sprintf(pipe_fd_str, "%d", pipes[i][1]);

            char *args[] = {"./journalist", journalist_id_str, pipe_fd_str, NULL};
            execvp(args[0], args);

            perror("execvp");
```

```
            exit(EXIT_FAILURE);
        } else {
            close(pipes[i][1]);
        }
    }

    int articles_published = 0;
    fd_set read_fds;
    int max_fd = 0;

    while (articles_published < num_articles_target) {
        FD_ZERO(&read_fds);
        for (int i = 0; i < num_journalists; i++) {
            FD_SET(pipes[i][0], &read_fds);
            if (pipes[i][0] > max_fd) {
                max_fd = pipes[i][0];
            }
        }

        int activity = select(max_fd + 1, &read_fds, NULL, NULL, NULL);
        if (activity < 0) {
            perror("select");
            exit(EXIT_FAILURE);
        }

        for (int i = 0; i < num_journalists; i++) {
            if (FD_ISSET(pipes[i][0], &read_fds)) {
                char buffer[ARTICLE_BUFFER_SIZE];
                ssize_t bytes_read = read(pipes[i][0], buffer, sizeof(buffer) - 1);

                if (bytes_read > 0) {
                    buffer[bytes_read] = '\0';
                    articles_published++;
                    printf("Editor: Published article! [%d/%d] -> \"%s\"\n", articles_published,
num_articles_target, buffer);
                    fflush(stdout);
                }
            }
        }
    }

    printf("Editor: Deadline met! Published %d articles. Shutting down newsroom.\n",
num_articles_target);
    fflush(stdout);

    for (int i = 0; i < num_journalists; i++) {
        kill(pids[i], SIGTERM);
    }

    for (int i = 0; i < num_journalists; i++) {
        wait(NULL);
    }
```

```
    for (int i = 0; i < num_journalists; i++) {
        close(pipes[i][0]);
    }


    printf("Editor: All journalists have exited. Newsroom closed.\n");

    return 0;
}
```

```
cse@cse:~/Desktop/2302CS11/Ass4_OS$ gcc editor.c -o editor
cse@cse:~/Desktop/2302CS11/Ass4_OS$ ./editor 2 4
Editor: Newsroom open. Waiting for 4 articles from 2 journalists.
Journalist 2, Researcher: Found topic "Story 1 from Journalist 2"
Journalist 2, Writer: Writing article on "Story 1 from Journalist 2"
Journalist 1, Researcher: Found topic "Story 1 from Journalist 1"
Journalist 2, Submitter: Submitting article.
Journalist 1, Writer: Writing article on "Story 1 from Journalist 1"
Journalist 1, Submitter: Submitting article.
Editor: Published article! [1/4] -> "Article on Story 1 from Journalist 2"
Editor: Published article! [2/4] -> "Article on Story 1 from Journalist 1"
Journalist 2, Researcher: Found topic "Story 2 from Journalist 2"
Journalist 2, Writer: Writing article on "Story 2 from Journalist 2"
Journalist 2, Submitter: Submitting article.
Journalist 1, Researcher: Found topic "Story 2 from Journalist 1"
Journalist 1, Writer: Writing article on "Story 2 from Journalist 1"
Editor: Published article! [3/4] -> "Article on Story 2 from Journalist 2"
Journalist 1, Submitter: Submitting article.
Editor: Published article! [4/4] -> "Article on Story 2 from Journalist 1"
Editor: Deadline met! Published 4 articles. Shutting down newsroom.
Journalist 1: Received termination signal. Exiting.
Journalist 2: Received termination signal. Exiting.
Editor: All journalists have exited. Newsroom closed.
```

journalist.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <signal.h>
#include <string.h>

#define ARTICLE_TOPIC_SIZE 100
#define ARTICLE_CONTENT_SIZE 256

typedef struct {
    pthread_mutex_t mutex;
    pthread_cond_t can_research;
    pthread_cond_t can_write;
    pthread_cond_t can_submit;

    char topic[ARTICLE_TOPIC_SIZE];
    char article[ARTICLE_CONTENT_SIZE];
```

```
    int topic_ready;
    int article_ready;

    int journalist_id;
    int article_count;
    int pipe_write_fd;
    volatile sig_atomic_t terminated;
} ThreadData;

ThreadData shared_data;

void handle_sigterm(int sig) {
    shared_data.terminated = 1;
}

void* researcher_thread(void* arg) {
    while (!shared_data.terminated) {
        pthread_mutex_lock(&shared_data.mutex);

        while (shared_data.topic_ready && !shared_data.terminated) {
            pthread_cond_wait(&shared_data.can_research, &shared_data.mutex);
        }

        if (shared_data.terminated) {
            pthread_mutex_unlock(&shared_data.mutex);
            break;
        }

        shared_data.article_count++;
        sprintf(shared_data.topic, "Story %d from Journalist %d", shared_data.article_count,
shared_data.journalist_id);
        printf("Journalist %d, Researcher: Found topic \"%s\"\n", shared_data.journalist_id,
shared_data.topic);
        fflush(stdout);

        shared_data.topic_ready = 1;
        pthread_cond_signal(&shared_data.can_write);
        pthread_mutex_unlock(&shared_data.mutex);

        sleep(1);
    }
    return NULL;
}

void* writer_thread(void* arg) {
    while (!shared_data.terminated) {
        pthread_mutex_lock(&shared_data.mutex);

        while (!shared_data.topic_ready && !shared_data.terminated) {
            pthread_cond_wait(&shared_data.can_write, &shared_data.mutex);
        }
```

```
        if (shared_data.terminated) {
            pthread_mutex_unlock(&shared_data.mutex);
            break;
        }

        sprintf(shared_data.article, "Article on %s", shared_data.topic);
        printf("Journalist %d, Writer: Writing article on \"%s\"\n", shared_data.journalist_id,
shared_data.topic);
        fflush(stdout);

        shared_data.topic_ready = 0;
        shared_data.article_ready = 1;
        pthread_cond_signal(&shared_data.can_submit);
        pthread_mutex_unlock(&shared_data.mutex);

        sleep(1);
    }
    return NULL;
}

void* submitter_thread(void* arg) {
    while (!shared_data.terminated) {
        pthread_mutex_lock(&shared_data.mutex);

        while (!shared_data.article_ready && !shared_data.terminated) {
            pthread_cond_wait(&shared_data.can_submit, &shared_data.mutex);
        }

        if (shared_data.terminated) {
            pthread_mutex_unlock(&shared_data.mutex);
            break;
        }

        printf("Journalist %d, Submitter: Submitting article.\n", shared_data.journalist_id);
        fflush(stdout);
        write(shared_data.pipe_write_fd, shared_data.article, strlen(shared_data.article));

        shared_data.article_ready = 0;
        pthread_cond_signal(&shared_data.can_research);
        pthread_mutex_unlock(&shared_data.mutex);

        sleep(1);
    }
    pthread_cond_broadcast(&shared_data.can_research);
    pthread_cond_broadcast(&shared_data.can_write);
    pthread_cond_broadcast(&shared_data.can_submit);
    return NULL;
}

int main(int argc, char *argv[]) {
    if (argc != 3) {
```

```
        fprintf(stderr, "Usage: %s <journalist_id> <pipe_fd>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    shared_data.journalist_id = atoi(argv[1]);
    shared_data.pipe_write_fd = atoi(argv[2]);
    shared_data.article_count = 0;
    shared_data.topic_ready = 0;
    shared_data.article_ready = 0;
    shared_data.terminated = 0;

    pthread_mutex_init(&shared_data.mutex, NULL);
    pthread_cond_init(&shared_data.can_research, NULL);
    pthread_cond_init(&shared_data.can_write, NULL);
    pthread_cond_init(&shared_data.can_submit, NULL);

    struct sigaction sa;
    sa.sa_handler = handle_sigterm;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;
    sigaction(SIGTERM, &sa, NULL);

    pthread_t researcher, writer, submitter;
    pthread_create(&researcher, NULL, researcher_thread, NULL);
    pthread_create(&writer, NULL, writer_thread, NULL);
    pthread_create(&submitter, NULL, submitter_thread, NULL);

    pthread_cond_signal(&shared_data.can_research);

    pthread_join(researcher, NULL);
    pthread_join(writer, NULL);
    pthread_join(submitter, NULL);

    pthread_mutex_destroy(&shared_data.mutex);
    pthread_cond_destroy(&shared_data.can_research);
    pthread_cond_destroy(&shared_data.can_write);
    pthread_cond_destroy(&shared_data.can_submit);

    close(shared_data.pipe_write_fd);
    printf("Journalist %d: Received termination signal. Exiting.\n", shared_data.journalist_id);
    fflush(stdout);

    return 0;
}
```

```
cse@cse:~/Desktop/2302CS11/Ass4_OS$ gcc journalist.c -o journalist
cse@cse:~/Desktop/2302CS11/Ass4_OS$ ./journalist 1 1
Journalist 1, Researcher: Found topic "Story 1 from Journalist 1"
Journalist 1, Writer: Writing article on "Story 1 from Journalist 1"
Journalist 1, Submitter: Submitting article.
Article on Story 1 from Journalist 1Journalist 1, Researcher: Found topic "Story 2 fro
m Journalist 1"
Journalist 1, Writer: Writing article on "Story 2 from Journalist 1"
Journalist 1, Submitter: Submitting article.
Article on Story 2 from Journalist 1Journalist 1, Researcher: Found topic "Story 3 fro
m Journalist 1"
```

```
cse@cse:~/Desktop/2302CS11/Ass4_OS$ .
/journalist 2 2
Journalist 2, Researcher: Found topic "Story 1 from Journalist 2"
Journalist 2, Writer: Writing article on "Story 1 from Journalist 2"
Journalist 2, Submitter: Submitting article.
Article on Story 1 from Journalist 2Journalist 2, Researcher: Found topic "Story 2 from
ournalist 2"
Journalist 2, Writer: Writing article on "Story 2 from Journalist 2"
Journalist 2, Submitter: Submitting article.
Article on Story 2 from Journalist 2Journalist 2, Researcher: Found topic "Story 3 from
ournalist 2"
```