OS-5

Interprocess Communication

`pipe(),msgget(),msgsnd(),msgrcv(),msgctl(),semget(),semop(),shmget()`
`shmat(),shmdt()`

1) Declare two character buffers of size 10 each, bufferToPipe, pipeToBuffer
Create an unnamed pipe using pipe system call
Create a child process.
The parent process asks for a 10 character string from the standard input (keyboard) and stores it in the character buffer bufferToPipe.
Then the parent writes this data onto the pipe and prints the message "data written into the pipe by parent". The child process reads from the pipe and fills the character buffer pipeToBuffer.
Then the child writes this data from pipeToBuffer onto the standard out (Screen).and prints the message "data read from the pipe by child".
The above repeats till the message read from the pipe is "over—exit" which the child displays and exits. Let the parent exit after the child exits (use wait system call).

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

#define BUFFER_SIZE 10

int main() {
    char bufferToPipe[BUFFER_SIZE];
    char pipeToBuffer[BUFFER_SIZE];
    int pipefd[2];

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

    pid_t pid = fork();

    if (pid == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
```

```
    }

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

        while (1) {
            ssize_t bytes_read = read(pipefd[0], pipeToBuffer,
BUFFER_SIZE);
            if (bytes_read == -1) {
                perror("read");
                exit(EXIT_FAILURE);
            }

            pipeToBuffer[bytes_read] = '\0';

            if (strcmp(pipeToBuffer, "over-exit") == 0) {
                printf("Data read from the pipe by child: %s\n",
pipeToBuffer);
                break;
            }

            printf("Data read from the pipe by child: %s\n",
pipeToBuffer);
        }

        close(pipefd[0]);
        exit(EXIT_SUCCESS);
    } else {
        close(pipefd[0]);

        while (1) {
            printf("Enter a 10 character string: ");
            fflush(stdout);

            fgets(bufferToPipe, BUFFER_SIZE, stdin);

            if (strlen(bufferToPipe) != BUFFER_SIZE) {
                printf("Please enter a 10 character string.\n");
                continue;
            }

            if (write(pipefd[1], bufferToPipe, BUFFER_SIZE) == -1) {
                perror("write");
                exit(EXIT_FAILURE);
            }
```

```
            printf("Data written into the pipe by parent.\n");

            if (strcmp(bufferToPipe, "over-exit\n") == 0) {
                break;
            }
        }

        close(pipefd[1]);

        wait(NULL);

        exit(EXIT_SUCCESS);
    }

    return 0;
}
```

2) Do the above problem with a named pipe created using mknod system call.

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

#define BUFFER_SIZE 10
#define FIFO_NAME "/tmp/myfifo"

int main() {
    char bufferToPipe[BUFFER_SIZE];
    char pipeToBuffer[BUFFER_SIZE];
    int fd;

    // Create the named pipe
    if (mknod(FIFO_NAME, S_IFIFO | 0666, 0) == -1) {
        perror("mknod");
        exit(EXIT_FAILURE);
    }

    pid_t pid = fork();
```

```
    if (pid == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
    }

    if (pid == 0) {
        if ((fd = open(FIFO_NAME, O_RDONLY)) < 0) {
            perror("open");
            exit(EXIT_FAILURE);
        }

        while (1) {
            ssize_t bytes_read = read(fd, pipeToBuffer, BUFFER_SIZE);
            if (bytes_read == -1) {
                perror("read");
                exit(EXIT_FAILURE);
            }

            pipeToBuffer[bytes_read] = '\0';

            if (strcmp(pipeToBuffer, "over-exit") == 0) {
                printf("Data read from the pipe by child: %s\n",
pipeToBuffer);
                break;
            }

            printf("Data read from the pipe by child: %s\n",
pipeToBuffer);
        }

        close(fd);
        exit(EXIT_SUCCESS);
    } else {
        if ((fd = open(FIFO_NAME, O_WRONLY)) < 0) {
            perror("open");
            exit(EXIT_FAILURE);
        }

        while (1) {
            printf("Enter a 10 character string: ");
            fflush(stdout);

            fgets(bufferToPipe, BUFFER_SIZE, stdin);

            if (strlen(bufferToPipe) != BUFFER_SIZE) {
```

```
                printf("Please enter a 10 character string.\n");
                continue;
            }

            if (write(fd, bufferToPipe, BUFFER_SIZE) == -1) {
                perror("write");
                exit(EXIT_FAILURE);
            }

            printf("Data written into the pipe by parent.\n");

            if (strcmp(bufferToPipe, "over-exit\n") == 0) {
                break;
            }
        }

        close(fd);

        wait(NULL);

        exit(EXIT_SUCCESS);
    }

    return 0;
}
```

3) Write a program to create the message queue and print its id.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int main() {
    key_t key;
    int msgid;

    if ((key = ftok("message_queue_key", 'B')) == -1) {
        perror("ftok");
        exit(EXIT_FAILURE);
    }
```

```
    if ((msgid = msgget(key, IPC_CREAT | 0666)) == -1) {
        perror("msgget");
        exit(EXIT_FAILURE);
    }

    printf("Message queue ID: %d\n", msgid);

    return 0;
}
```

4) Write a program to destroy the message queue created in above program

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int main() {
    key_t key;
    int msgid;

    if ((key = ftok("message_queue_key", 'B')) == -1) {
        perror("ftok");
        exit(EXIT_FAILURE);
    }

    if ((msgid = msgget(key, IPC_CREAT | 0666)) == -1) {
        perror("msgget");
        exit(EXIT_FAILURE);
    }

    if (msgctl(msgid, IPC_RMID, NULL) == -1) {
        perror("msgctl");
        exit(EXIT_FAILURE);
    }

    printf("Message queue destroyed successfully.\n");

    return 0;
}
```

5) Write a program which creates a message queue and sends lines of text into queue.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define MAX_TEXT_SIZE 100

struct msg_buffer {
    long msg_type;
    char msg_text[MAX_TEXT_SIZE];
};

int main() {
    key_t key;
    int msgid;
    struct msg_buffer message;

    if ((key = ftok("message_queue_key", 'B')) == -1) {
        perror("ftok");
        exit(EXIT_FAILURE);
    }

    if ((msgid = msgget(key, IPC_CREAT | 0666)) == -1) {
        perror("msgget");
        exit(EXIT_FAILURE);
    }

    message.msg_type = 1;

    printf("Enter lines of text to send to the message queue (type 'exit'
to stop):\n");

    while (1) {
        fgets(message.msg_text, MAX_TEXT_SIZE, stdin);

        if (strcmp(message.msg_text, "exit\n") == 0) {
            break;
        }

        if (msgsnd(msgid, (void *)&message, MAX_TEXT_SIZE, 0) == -1) {
            perror("msgsnd");
            exit(EXIT_FAILURE);
```

```
        }
    }

    printf("Lines of text sent to the message queue.\n");

    return 0;
}
```

6) Write a program which receives messages from above created queue.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define MAX_TEXT_SIZE 100

struct msg_buffer {
    long msg_type;
    char msg_text[MAX_TEXT_SIZE];
};

int main() {
    key_t key;
    int msgid;
    struct msg_buffer message;

    if ((key = ftok("message_queue_key", 'B')) == -1) {
        perror("ftok");
        exit(EXIT_FAILURE);
    }

    if ((msgid = msgget(key, 0666)) == -1) {
        perror("msgget");
        exit(EXIT_FAILURE);
    }

    printf("Messages received from the message queue:\n");

    while (1) {
        if (msgrcv(msgid, (void *)&message, MAX_TEXT_SIZE, 0, 0) == -1) {
            perror("msgrcv");
```

```
            exit(EXIT_FAILURE);
        }

        printf("%s", message.msg_text);

        if (strcmp(message.msg_text, "exit\n") == 0) {
            break;
        }
    }

    return 0;
}
```