**Write a program for producer-consumer paradigm, both processes are running concurrently.**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/wait.h>

#include <sys/mman.h>

#define BUFFER\_SIZE 10

typedef struct {

int buffer[BUFFER\_SIZE];

int index;

} shared\_data;

void producer(shared\_data \*data) {

int i, value;

for (i = 0; i < 20; i++) {

value = rand() % 100;

while (data->index == BUFFER\_SIZE) {

printf("Buffer is full, producer waiting...\n");

sleep(1);

}

data->buffer[data->index++] = value;

printf("Producer added value %d to buffer\n", value);

}

}

void consumer(shared\_data \*data) {

int i, value;

for (i = 0; i < 20; i++) {

while (data->index == 0) {

printf("Buffer is empty, consumer waiting...\n");

sleep(1);

}

value = data->buffer[--data->index];

printf("Consumer removed value %d from buffer\n", value);

}

}

int main() {

int fd, result;

shared\_data \*data;

fd = shm\_open("/myshm", O\_CREAT | O\_RDWR, 0666);

if (fd == -1) {

perror("shm\_open");

exit(1);

}

result = ftruncate(fd, sizeof(shared\_data));

if (result == -1) {

perror("ftruncate");

exit(1);

}

data = mmap(NULL, sizeof(shared\_data), PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

if (data == MAP\_FAILED) {

perror("mmap");

exit(1);

}

data->index = 0;

pid\_t pid = fork();

if (pid == -1) {

perror("fork");

exit(1);

} else if (pid == 0) {

consumer(data);

} else {

producer(data);

wait(NULL);

result = shm\_unlink("/myshm");

if (result == -1) {

perror("shm\_unlink");

exit(1);

}

}

return 0;

}

