## **Semaphore.h**

#include <stdio.h>

#include <sys/ipc.h>

#include <sys/sem.h>

# // Create or get a group of nsems semaphores with given key

# // Returns semaphore group ID

int create\_sm(int nsems, key\_t key) {

int semid = semget(key, nsems, IPC\_CREAT | IPC\_EXCL | 0666);

if (semid < 0) {

# // If creation fails, get existing semaphore group

semid = semget(key, nsems, 0);

printf("Semaphor group already exists with id (%d)\n", semid);

} else

printf("New semaphore group created with id (%d)\n", semid);

return semid;

}

# // Initialize a semaphore in group semid with given value

void init\_sm(int semid, int semnum, int val) {

if (semctl(semid, semnum, SETVAL, val) < 0) {

perror("Init semctl failed\n");

} else {

printf("Init semctl successful\n");

}

}

# // P operation (wait/decrement) on semaphore

void p(int semid, int semnum) {

struct sembuf sb;

sb.sem\_num = semnum; // Semaphore number in group

sb.sem\_op = -1; // Decrement operation

sb.sem\_flg = SEM\_UNDO; // Auto cleanup if process dies

if (semop(semid, &sb, 1) < 0) {

perror("P semop failed\n");

} else {

printf("P semop executed successfully\n");

}

}

# // V operation (signal/increment) on semaphore

void v(int semid, int semnum) {

struct sembuf sb;

sb.sem\_num = semnum; // Semaphore number in group

sb.sem\_op = 1; // Increment operation

sb.sem\_flg = SEM\_UNDO; // Auto cleanup if process dies

if (semop(semid, &sb, 1) < 0) {

perror("V semop failed\n");

} else {

printf("V semop executed successfully\n");

}

}

# // Z operation (wait for zero) on semaphore

void z(int semid, int semnum) {

struct sembuf sb;

sb.sem\_num = semnum; // Semaphore number in group

sb.sem\_op = 0; // Wait for zero operation

sb.sem\_flg = 0; // No special flags needed

if (semop(semid, &sb, 1) < 0) {

perror("Z semop failed\n");

} else {

printf("Z semop executed successfully\n");

}

}

# // Remove/destroy a semaphore group

void sem\_destroy(int semid) {

if (semctl(semid, 0, IPC\_RMID) < 0) {

perror("semctl IPC\_RMID failed\n");

} else {

printf("Semaphore set removed successfully.\n");

}

}

## **Create.c:**

#include "semaphore.h"

#include <stdio.h>

#include <stdlib.h>

#include <sys/shm.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

# // Define shared memory structure for producer-consumer

typedef struct shared\_data {

int producer\_index; // Index where producer will write

int consumer\_index; // Index where consumer will read

int buffer[10]; // Circular buffer of size 10

} shared\_data;

int main() {

# // Generate unique key for shared memory using current directory

key\_t key = ftok(".", 'b');

# // Create shared memory segment

# // IPC\_CREAT: create if doesn't exist

# // IPC\_EXCL: fail if already exists

# // 0666: read/write permissions for all

int shmid = shmget(key, sizeof(shared\_data), IPC\_CREAT | IPC\_EXCL | 0666);

# // Handle shared memory creation/access

if (shmid < 0) {

// If creation failed, try to get existing segment

shmid = shmget(key, sizeof(shared\_data), 0);

printf("Le segment de mémoire partagée existe déjà avec l'ID : %d\n",

shmid);

} else {

printf("Nouveau segment de mémoire partagée créé avec l'ID : %d\n", shmid);

}

# // Attach shared memory segment to process address space

shared\_data \*sd = shmat(shmid, NULL, 0);

if (sd == (void \*)-1) {

perror("Échec de shmat");

exit(EXIT\_FAILURE);

}

# // Initialize shared memory indices to 0

sd->producer\_index = 0;

sd->consumer\_index = 0;

printf("Indexes initialisés (Producteur : %d, Consommateur : %d)\n",

sd->producer\_index, sd->consumer\_index);

# // Create and initialize semaphores:

# // semid[0]: counts empty slots (initial=10)

# // semid[1]: counts full slots (initial=0)

# // semid[2]: mutex for producers (initial=1)

# // semid[3]: mutex for consumers (initial=1)

int semid = create\_sm(4, key);

init\_sm(semid, 0, 10); // Empty slots counter

init\_sm(semid, 1, 0); // Full slots counter

init\_sm(semid, 2, 1); // Producer mutex

init\_sm(semid, 3, 1); // Consumer mutex

return 0;

}

## **Produce.c:**

#include "semaphore.h"

#include <stdlib.h>

#include <sys/shm.h>

#include <unistd.h>

# // Define shared memory structure for producer-consumer

typedef struct shared\_data {

int producer\_index; // Write position in buffer

int consumer\_index; // Read position in buffer

int buffer[10]; // Circular buffer

} shared\_data;

int main() {

# // Generate unique key for shared memory

key\_t key = ftok(".", 'b');

# // Try to create shared memory segment

int shmid = shmget(key, sizeof(shared\_data), IPC\_CREAT | IPC\_EXCL | 0666);

if (shmid < 0) { // Segment already exists

shmid = shmget(key, sizeof(shared\_data), 0);

printf("Le segment de mémoire partagée existe déjà avec l'ID : %d\n",

shmid);

} else {

printf("Nouveau segment de mémoire partagée créé avec l'ID : %d\n", shmid);

}

# // Attach shared memory to process address space

shared\_data \*sd = shmat(shmid, NULL, 0);

if (sd == (void \*)-1) {

perror("Échec de shmat");

exit(EXIT\_FAILURE);

}

# // Get access to semaphores

int semid = create\_sm(4, key);

int value;

# // Infinite production loop

while (1) {

# // Skip if current buffer position is not empty

if (sd->buffer[sd->producer\_index] != 0) {

continue;

}

# // Generate random value between 0-9

value = rand() % 10;

printf("Valeur produite : %d\n", value);

# // Semaphore operations for synchronization

p(semid, 0); // Wait for empty slot

p(semid, 2); // Get producer mutex

# // Write to buffer and update index

sd->buffer[sd->producer\_index] = value;

sd->producer\_index = (sd->producer\_index + 1) % 10; // Circular increment

v(semid, 2); // Release producer mutex

v(semid, 1); // Signal one slot is full

# // Display current buffer state

printf("Buffer = [%d, %d, %d, %d, %d, %d, %d, %d, %d, %d]\n", sd->buffer[0],

sd->buffer[1], sd->buffer[2], sd->buffer[3], sd->buffer[4],

sd->buffer[5], sd->buffer[6], sd->buffer[7], sd->buffer[8],

sd->buffer[9]);

# // Wait 5 seconds before next production

sleep(5);

}

return 0;

}

## **Consumer.c:**

#include "semaphore.h"

#include <stdlib.h>

#include <sys/shm.h>

#include <unistd.h>

# // Define shared memory structure (same as producer and create)

typedef struct shared\_data {

int producer\_index; // Producer's write position

int consumer\_index; // Consumer's read position

int buffer[10]; // Circular buffer

} shared\_data;

int main() {

# // Generate unique key for shared memory

key\_t key = ftok(".", 'b');

# // Try to create/get shared memory segment

int shmid = shmget(key, sizeof(shared\_data), IPC\_CREAT | IPC\_EXCL | 0666);

if (shmid < 0) { // Segment exists

shmid = shmget(key, sizeof(shared\_data), 0);

printf("Le segment de mémoire partagée existe déjà avec l'ID : %d\n",

shmid);

} else {

printf("Nouveau segment de mémoire partagée créé avec l'ID : %d\n", shmid);

}

# // Attach shared memory to process address space

shared\_data \*sd = shmat(shmid, NULL, 0);

if (sd == (void \*)-1) {

perror("Échec de shmat");

exit(EXIT\_FAILURE);

}

# // Get access to semaphores

int semid = create\_sm(4, key);

int value;

# // Infinite consumption loop

while (1) {

# // Skip if current buffer position is empty

if (sd->buffer[sd->consumer\_index] == 0) {

continue;

}

# // Semaphore operations for synchronization

p(semid, 1); // Wait for full slot

p(semid, 3); // Get consumer mutex

# // Read from buffer and mark slot as empty

value = sd->buffer[sd->consumer\_index];

sd->buffer[sd->consumer\_index] = 0;

sd->consumer\_index = (sd->consumer\_index + 1) % 10; // Circular increment

v(semid, 3); // Release consumer mutex

v(semid, 0); // Signal one slot is empty

# // Display consumed value and buffer state

printf("Consommation de la valeur : %d, à l'index %d\n", value,

sd->consumer\_index);

printf("Tampon = [%d, %d, %d, %d, %d, %d, %d, %d, %d, %d]\n",

sd->buffer[0], sd->buffer[1], sd->buffer[2], sd->buffer[3],

sd->buffer[4], sd->buffer[5], sd->buffer[6], sd->buffer[7],

sd->buffer[8], sd->buffer[9]);

# // Wait 5 seconds before next consumption

sleep(5);

}

return 0;

}