ALGORITHM OF IPC THROUGH SHARED MEMORY

Description

We develop Algorithm of two programs here that illustrate the passing of a simple piece of memory (a string) between the processes if running simultaneously:

sharedmem1.c

--Attaches itself to the created shared memory portion and uses the string.

Sharedmem2.c

--simply creates the string from user input and shared memory portion.

Algorithm of sharedmem2.c (Server Side Program)

STEP – 1. In a C program first of all include all header files such as unistd.h , stdlib.h , stdio.h , string.h

Now include header file sys/shm.h to access shared memory system calls

STEP – 2 Struct shared_use_st --- creating a shared_use_st of type struct

int data_available --- variable which indicates data is available

char message[Text_SZ] --- char array to hold the input string

main(){

process_running = TRUE

void *shared_memory = (void *)0

STEP – 3 shared_use_st *shared stuff --- structure which acts like a shared memory

char buffer[BUFSIZ]

int shmid --- varaiable for allocation of shared memory

shmid = shmget((key_t)1234, sizeof(struct shared_use_st) ,0666 | IPC_CREAt) --- Allocation of Shared Memory

on failure shmget () returns -1

STEP - 4 Attachment of segement using shmat() system call

Shared_memory = shmat(shmid,(void *)0, 0)

On failure shmat() returns -1

Shared_stuff = (struct shared_use_st *)shared_memory --- on successful attachment of segment

Performing write operation

While(process_running)

While(shared_stuff->data_available == TRUE)

Sleep(1)

STEP – 5 Server creates the string by asking the user an input

STEP – 6 Reads the input and stores it in buffer

STEP – 7 Writes it into the char array message[]

Check if data is available

STEP – 8 Terminate the loop by using string "end"

Process_running = FALSE --- indicates that the data is written and sets the flag

STEP – 9 Now Detach segment using shmdt() system call

Exit(EXIT_SUCCESS)

} ---end of the server program

Algorithm of sharedmem1.c (Client Side Program)

STEP – 1 In a C program first of all include all header files such as unistd.h , stdlib.h , stdio.h , string.h

Now include header file sys/shm.h to access shared memory system calls.

Now include header file sys/shm.h to access shared memory system calls.

STEP – 2 Struct shared_use_st --- creating a shared_use_st of type struct

int data_available --- variable which indicates data is available

char message[Text_SZ] --- char array to hold the input string

main(){

process_running = TRUE

void *shared_memory = (void *)0

STEP – 3 shared_use_st *shared stuff --- structure which acts like a shared memory

int shmid --- varaiable for allocation of shared memory

STEP – 4 shmid = shmget((key_t)1234, sizeof(struct shared_use_st) ,0666 | IPC_CREAt) --- Allocation of Shared Memory

on failure shmget () returns -1

STEP – 5 Attachment of segement using shmat() system call

Shared_memory = shmat(shmid,(void *)0, 0)

On failure shmat() returns -1

STEP – 6 If shared_memory == (void *)-1

The exit(EXIT_FAILURE)

Shared_stuff = (struct shared_use_st *) shared_memory

Shared_stuff->data_available = FALSE

While process_running

If data available --- means data is available to read

STEP – 7 Print data

Sleep(rand() %4)

Shared_stuff->data_available = FALSE --- clears the flag to show it has read the data

STEP – 8 Now Terminate the loop with the help of string “end”

STEP – 9 At the end detach the shared memory with the help of shmdt() system call or function

On failure shmdt() returns -1

Now Deallocating the shared memory with the help of shmctl() system call

Which on failure returns -1

Exit(EXIT_SUCCESS)

} --- End of Client Process