System Programming

Module 8

EGCI 252
System Programming

Computer Engineering Department
Faculty of Engineering

Shared Memory

- Normally, the Unix kernel prohibits one process from accessing (reading, writing) memory belonging to another process
- Sometimes, however, this restriction is inconvenient
- At such times, System V IPC Shared Memory can be created to specifically allow on process to read and/or write to memory created by another process

Advantages of Shared Memory

- Random Access
  - you can update a small piece in the middle of a data structure, rather than the entire structure
- Efficiency
  - unlike message queues and pipes, which copy data from the process *into* memory within the kernel, shared memory is directly accessed
  - Shared memory resides in the user process memory, and is then shared among other processes

Disadvantages of Shared Memory

- No automatic synchronization as in pipes or message queues (you have to provide any synchronization). Synchronize with *semaphores* or signals.
- You must remember that pointers are only valid within a given process. Thus, pointer offsets cannot be assumed to be valid across inter-process boundaries. This complicates the sharing of linked lists or binary trees.

Creating Shared Memory

int shmget(key_t key, size_t size, int shmflg);

- key is either a number or the constant IPC_PRIVATE (man ftok)
- a shmid is returned
- key_t ftok(const char * path, int id) will return a key value for IPC usage
- size is the size of the shared memory data
- shmflg is a rights mask (0666) OR'd with one of the following:
  - IPC_CREAT will create or attach
  - IPC_EXCL creates new or it will error if it exists

Attaching Shared Memory

- After obtaining a shmid from shmget(), you need to *attach* or map the shared memory segment to the address space of a process:

  void * shmat(int shmid, void * shmaddr, int shmflg)

  - shmid is the id returned from shmget()
  - shmaddr is the shared memory segment address. Set this to NULL and let the system handle it.
  - shmflg is one of the following (usually 0):
    - SHM_RDONLY sets the segment readonly
    - SHM_RND sets page boundary access
    - SHM_SHARE_MMU set first available aligned address

Detaching Shared Memory

- In order to detach the shared memory from the current process, the shmdt function must be issued.

  void * shmdt(void * first_byte_addr)
- first_byte_addr is a pointer to the address returned by shmat() function.

Shared Memory Control

```
struct shmid_ds {
 int shm_segsz;        /* size of segment in bytes */
  __time_t shm_atime;                  /* time of last shmat command */
  __time_t shm_dtime;                  /* time of last shmdt command */
  ...
  unsigned short int __shm_npages;    /* size of segment in pages */
  msgqnum_t shm_nattach;          /* number of current attaches */
    ...                          /* pids of creator and last shmop */
};
```

- int shmctl(int shmid, int cmd, struct shmid_ds * buf);
- cmd can be one of:
  - IPC_RMID destroy the memory specified by shmid
  - IPC_SET set the uid, gid, and mode of the shared mem
  - IPC_STAT get the current shmid_ds struct for the queue

Shared Memory - Example

```
//Consumer.c
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define MEM_SIZE 4096

struct shm_st
{
    int written;
    char data[BUFSIZ];
}

int main()
{
    int running = 1, shmID;
    void *sh_mem = NULL;
    struct shm_st *sh_area;

    srand((unsigned int) getpid());
    shmid = shmget((key_t) 1234, MEM_SIZE,
                                0666 | IPC_CREAT);
    if (shmid == -1)
    {
        fprintf(stderr, “shmget failed\n”);
        exit(EXIT_FAILURE);
    }

    sh_mem = shmat(shmID, NULL, 0);
    if (sh_mem == (void *) -1)
    {
        fprintf(stderr, “shmat failed\n”);
        exit(EXIT_FAILURE);
    }

    printf(“Memory attached at %X\n”, sh_mem);
    sh_area = (struct shm_st *) sh_mem;
    sh_area->written = 0;
```

Shared Memory - Example (Cont.)

```
while (running)
{
    if (sh_area->written)
    {
        printf("Data written in shared memory: %s\n", sh_area-> data);
        sh_area->written = 0;
        if (strncmp(sh_area->data, "end", 3) == 0)
            running = 0;
    }
    sleep(rand() % 4);
  }

  if (shmdt(sh_mem) == -1 || shmctl(shmid, IPC_RMID, 0) == -1)
  {
      fprintf(stderr, "shmdt or shmctl failed\n");
      exit(EXIT_FAILURE);
  }

   exit(EXIT_SUCCESS);
}
```

Shared Memory – Example (Cont.)

```
//Producer.c
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define MEM_SIZE 4096

struct shm_st
{
    int written;
    char data[BUFSIZ];
}

int main()
{
    int running = 1, shmID;
    void *sh_mem = NULL;
    struct shm_st *sh_area;
```

```
char buffer[BUFSIZ];

shmid = shmget((key_t) 1234, MEM_SIZE,
                              0666 | IPC_CREAT);
if (shmid == -1)
{
    fprintf(stderr, “shmget failed\n”);
    exit(EXIT_FAILURE);
}

sh_mem = shmat(shmID, NULL, 0);
if (sh_mem == (void *) -1)
{
    fprintf(stderr, “shmat failed\n”);
    exit(EXIT_FAILURE);
}

printf(“Memory attached at %X\n”, sh_mem);
sh_area = (struct shm_st *) sh_mem;
```

Shared Memory – Example (Cont.)

```
while (running)
  {
       while (sh_area->written)
       {
          sleep(1); printf(“Waiting…\n”);
       }
       printf(“Enter data: ”);
       fgets(buffer, BUFSIZ, stdin);
       strcpy(sh_area->data, buffer);
       sh_area->written = 1;
       if (strncmp(buffer, “end”, 3) == 0)
           running = 0;
   }

   if (shmdt(sh_mem) == -1)
   {
       fprintf(stderr, “shmdt failed\n”);
       exit(EXIT_FAILURE);
   }
   exit(EXIT_SUCCESS);
}
```

Assignment

- Write a simple chat program using a shared memory
- Requirements:
  - The program's name must be "schat.c"
  - The program takes one command line argument (i.e., 1 or 2 to indicated the type of messages)
- Create a shared memory with the key value of 21930
- Must be able to concurrently send and receive any messages between two "schat" processes by using the shared memory.
- Use the word "end chat" as a command to end the chat program

End of Module