Computer Science & Engineering Department
I. I. T. Kharagpur

Operating System: CS33007
3rd Year CSE: 5th Semester (*Autumn 2006 - 2007*)
Lecture VIII (Linux System Calls IV)

Goutam Biswas *Date:* 22nd-23rd August, 2006

1 Requesting for Memory

Here we shall consider two sets of system calls requesting for memory. The first one is request for data memory and the second one is request for the shared memory for communicating between processes.

The command `pmap` shows the memory map of a process.

```
/*************************************
 * Used to see the virtual memory
 * regions of a process using
 * $ pmap -x <pid>
 *************************************/
#include <stdio.h>

int main(){
    printf("Virtual memory\n") ;
    while(1) ;
    printf("Over\n") ;
    return 0 ;
}
```

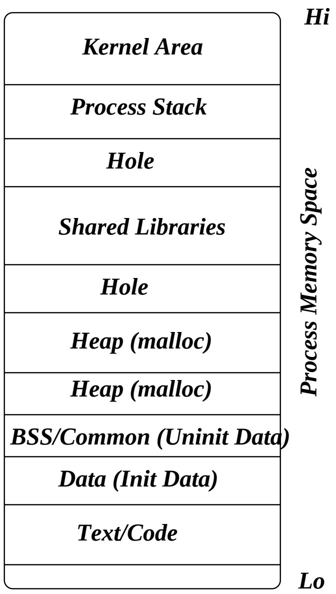
Compile and run this code in the background and type the command `pmap -x` <`pid`>

This data is also available from `/proc/`<`pid`>`/maps`. Different regions of memory are

1.1 Data Memory Request

The system call `int brk(void *end)` (45) changes the data segment of the program. The C library wrapper `void *sbrk(int inc)` increases the size of the data segment to `inc` number of bytes and returns the starting address of the new area. `sbrk(0)` returns the current location of program break.

```
/**********************************************
 * Program break
 *               sbrkProg1.c
 * Use: ps -A
 *      cat /proc/<pid>/maps
 *********************************************/
```




```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main() {
        char *p ;

        p = (char *)sbrk(0) ;
        printf("Prog. Brk. %p - %u\n", p, (unsigned) p) ;
        while(1) ;
return 0 ;
}
```

This part of the memory is not available.

```
/*********************************************
 * This area is not available
 *               sbrkProg1a.c
 * Use: ps -A
 *      cat /proc/<pid>/maps
 *********************************************/

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main() {
        char *p ;

        p = (char *)sbrk(0) ;
        printf("Prog. Brk. %p - %u\n", p, (unsigned) p) ;
        *p = 'a' ;
        while(1) ;
return 0 ;
}
```

We do a `malloc()` and the area will be available. What will be available is more than what we malloc().

```
/*********************************************
 * The area is made available by malloc
 *               sbrkProg1b.c
 * Use: ps -A
 *      cat /proc/<pid>/maps
 *********************************************/
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main() {
        char *p, *q ;

        p = (char *)sbrk(0) ;
        printf("Prog. Brk. %p - %u\n", p, (unsigned) p) ;
        q = (char *)malloc(1) ;
        p = (char *)sbrk(0) ;
        printf("New Prog. Brk. %p - %u\n", p, (unsigned) p) ;
        while(q < p) *q++ = 'A' ;

        while(1) ;
return 0 ;
```

```
}
```

If the condition of the `while` is made $q <= p$, there will be memory fault.

1.2 Shared memory

Two or mor processes can request the OS for shared memory area for communication. Following example shows how two processes can share a memory area.

```
/***************************************************
 * Creating a shared memory segment and attaching it
 * to the logical address space.  sharedMem1.c
 * $ cc -Wall sharedMem1.c
 * $ ./a.out w
 * $ ./a.out r
 * *************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#define SIZE 4

int main(int count, char *vect[]) {
int shmID, *p ;

if(count < 2) {
printf("No 2nd argument\n") ;
exit(0) ;
}
shmID = shmget(ftok("/home/projector", 1234), SIZE, IPC_CREAT | 0777);
if(shmID == -1) {
perror("Error in shmget") ;
exit(0) ;
}
p = (int *) shmat(shmID, 0, 0777) ;
if(strcmp(vect[1], "w") == 0) {
printf("Enter an integer: ") ;
scanf("%d", p) ;              // Write data
}
else printf("The data is %d\n", *p) ; // Read data
shmdt(p) ;
        // The shared memory segment remains in the system
        // $ ipcs
        // $ ipcrm -m<number>
return 0 ;
}
```

The next code shows the size of the shared memory.

```
/*****************************************************
 * Size of the shared memory segment
 * $ cc -Wall sharedMem2.c
 * ***************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define PAGE_SIZE 4096

int main() {
int shmID, *p, i, n ;
struct shmid_ds buff ;

shmID = shmget(IPC_PRIVATE, 4, IPC_CREAT | 0777);
if(shmID == -1) {
perror("Error in shmget") ;
exit(0) ;
}
p = (int *) shmat(shmID, 0, 0777) ;
        printf("\tShared memory address: %p\n", p) ;
n = PAGE_SIZE/sizeof(int) ;
        // sleep(20) ;  // To see the /proc/<pid>/maps
for(i=0; i<n; ++i) p[i] = i ;
// p[i] = i ;   // Gives Segmentation fault
        for(i=0; i<n; ++i) printf("%d ", p[i]) ;
        printf("\n") ;

shmdt(p) ;
shmctl(shmID, IPC_RMID, &buff) ;

return 0 ;
}
```

Race condition in shared memory access. Produce-consumer problem.
Bounded queue:

```
/****** queue.h **********/
          #ifndef _QUEUE_H
          #define _QUEUE_H

          #define MAX 5
```

```
          #define ERROR 1
          #define OK 0

          typedef struct {
              int data[MAX] ;
              int front , rear, count ;
          } queue ;

          /* Queue may contain MAX data.*/

           void initQ(queue *) ;
           int addQ(queue * , int) ;
           int deleteQ(queue *);
           int frontQ(queue *, int *) ;
   int isEmptyQ(queue *) ;
   int isFullQ(queue *) ;
          #endif

/****** queue.c *******/
#include "queue.h"
#include <unistd.h>
#include <stdlib.h>

void initQ(queue *q) {
q -> front = q -> rear = 0 ;
q -> count = 0 ;
}

int addQ(queue *q, int n) {
int i, lim, m ;

  if(isFullQ(q)) return ERROR ;
  q -> rear = (q -> rear + 1 ) % MAX ;
  q -> data[q -> rear] = n ;
m = q -> count ;
m = m + 1 ;
// lim = rand()%10000000 ; // Delay
        // for(i=1; i<= lim; ++i) ; //
q -> count = m ;
  return OK ;
}

int isEmptyQ(queue *q) {
return q -> count == 0 ;
}

int isFullQ(queue *q) {
```

```
  return q -> count == MAX ;
}

int deleteQ(queue *q) {
int n, i, lim ;

  if(isEmptyQ(q)) return ERROR ;
  q -> front = (q -> front + 1 ) % MAX ;
n = q -> count ;
n = n - 1 ;
// lim = rand()%10000000 ; // Delay
        // for(i=1; i<= lim; ++i) ; //
q -> count = n ;
  return OK ;
}

int frontQ(queue *q , int *v) {
  if(isEmptyQ(q)) return ERROR ;
  *v = q -> data[(q -> front + 1) % MAX] ;
  return OK ;
}
```

The shared memory code

```
/*************************************************************
 * Race condition
 * Producer-Consumer Problem on shared memory between
 * processes.
 * $ cc -Wall sharedProdCons1.c queue.o
 * ***********************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/wait.h>
#include <unistd.h>
#include "queue.h"

void producer(queue *) ;
void consumer(queue *) ;

int main() {
int shmID, chID1, chID2, status ;
struct shmid_ds buff ;
```

```
shmID = shmget(IPC_PRIVATE, sizeof(queue), IPC_CREAT | 0777);
if(shmID == -1) {
printf("Error in shmget\n") ;
exit(0) ;
}
if((chID1 = fork()) != 0) { // Parent
if((chID2 = fork()) != 0) { // Parent now
queue *qP ;

qP = (queue *) shmat(shmID, 0, 0777) ;
initQ(qP) ;

waitpid(chID1, &status, 0) ;
waitpid(chID2, &status, 0) ;

shmdt(qP) ;
shmctl(shmID, IPC_RMID, &buff) ;
}
else { // Child 2
queue *qP ;

qP = (queue *) shmat(shmID, 0, 0777) ;
consumer(qP) ;
shmdt(qP) ;
}
}
else { // Child I
queue *qP ;

qP = (queue *) shmat(shmID, 0, 0777) ;
producer(qP) ;
shmdt(qP) ;
}

return 0 ;
}

void producer(queue *qP){
int count = 1, added = 1 ;
        while(1) {
                int data, err;

if(added) {
data = rand() ;
        added = 0 ;
}
```

```
        err = addQ(qP, data) ;
if(err == OK) {
added = 1 ;
        printf("Produced Data %d: %d\n", count++, data) ;
}
        }
}

void consumer(queue *qP) {
        int count = 1 ;
        while(1) {
                int data, deleted ;

        frontQ(qP, &data) ;
        deleted = deleteQ(qP) ;
if(deleted == OK)
        printf("\t\t\tConsumed Data %d: %d\n", count++, data) ;
        }
}
```

References

[1]