Austin Brown

CPE 434-01

2/7/2021

Lab 4

The purpose of this lab is to write to programs that share a memory space. This means that we have multiple processes that can communicate with each other. We use the shmget(), shmat(), shmctl(), and shmdt() functions to accomplish this.

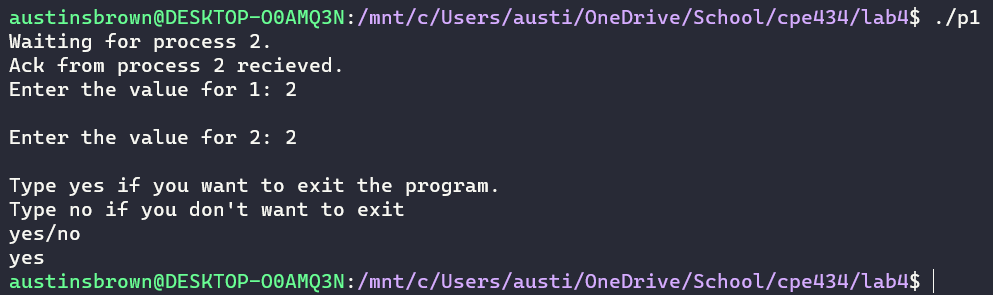
The shmget() function is used to create a shared memory space. We pass in the key of 1234, the size of the space that we want, and a flag. The flag is IPC\_CREAT | 0666. This gives the process read write permissions to the space.

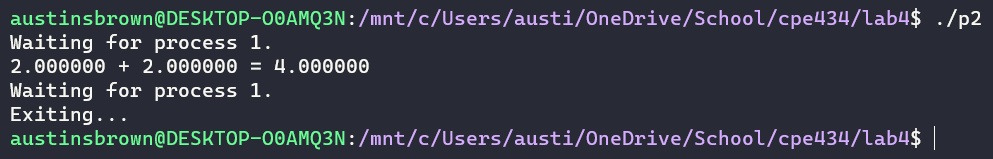
The shmat() function attaches the address space of the process to the shared memory space. The first argument to be passed is the identifier of the space. You then pass the attaching address. It is set to 0. The flag is also set to 0.

The shmdt() function detaches a shared memory process. We simply pass the address of the shared space into the function. This function is used in both process 1 and 2.

The shmctl() performs an operation on the shared space. We pass in the identifier first and then the command. We set the buffer argument to null. This destroys the memory segment once the final process is detached from it.

**Outputs:**

****

****

**Code:**

|  |
| --- |
| typedef *struct* *SharedData*  {  *float* value1, value2;  *float* sum;    */\**  *2 if waiting on process 2,*  *1 if wating on process 1,*  *-1 if done*  *\*/*  *int* flag;  }*SharedData*;  #define KEY (*key\_t*)(1244)  #define STRUCTSIZE sizeof(*SharedData*) |

|  |
| --- |
| #include <stdio.h>  #include <sys/types.h>  #include <string.h>  #include <sys/ipc.h>  #include <sys/shm.h>  #include <stdlib.h>  #include "header.h"  *int* main()  {  *int* identifier; *// holds a pid*  *int* i; *// generic counter*  *SharedData* \*control; *// info struct*  *char* buffer[255]; *// holds the user input*      strcpy(buffer, "NO");    */\**  *allocates shared mem segment*  *key of 1234 is passed in as first arg*  *size of the struct is passed in as second arg*  *IPC\_CREAT | 0666 sets it to read write*  *\*/*      if((identifier = shmget(KEY, STRUCTSIZE, IPC\_CREAT | 0666)) < 0) *// check for errors*      {          printf("An error has occured. Could not create memory segment.\n");          exit(-1);      }      if ((control = (*SharedData*\*) shmat(identifier, 0, 0)) <= (*SharedData*\*)(0))      {          printf("Error: Failed to attach address space to shared memory. Terminating...\n");          exit(1);      }        control->flag = 2; *// waiting on process 2*      printf("Waiting for process 2.\n");      while (control->flag == 2);      printf("Ack from process 2 recieved.\n");      while(control->flag != -1)      {          if(strcasecmp(buffer, "Y") == 0 || strcasecmp(buffer, "YES") == 0) *// exit the program?*          {             control->flag = -1;          }          else if(strcasecmp(buffer, "N") == 0 || strcasecmp(buffer, "NO") == 0) *// repeat the process*          {              printf("Enter the value for 1: ");              scanf("%s", buffer);              control->value1 = atoi(buffer);              printf("\nEnter the value for 2: ");              scanf("%s", buffer);              control->value2 = atoi(buffer);              control->flag = 1;          }          if(control->flag != -1) *// exit the program.*          {              while(control->flag != 0);              printf("\nType yes if you want to exit the program.\n");              printf("Type no if you don't want to exit\n");              fgets(buffer, 255, stdin);              buffer[strcspn(buffer, "\n")] = 0;              while(!(strcasecmp(buffer, "Y") == 0 || strcasecmp(buffer, "YES") == 0) && !(strcasecmp(buffer, "N") == 0 || strcasecmp(buffer, "NO") == 0))              {                  printf("yes/no\n");                  fgets(buffer, 255, stdin);                  buffer[strcspn(buffer, "\n")] = 0;              }          }      }      shmdt(control); *// detach shared memory space*      shmctl(identifier, IPC\_RMID, NULL); *// mark shared memor space for deletion*      return 0;  } |

|  |
| --- |
| #include <stdio.h>  #include <sys/types.h>  #include <string.h>  #include <sys/ipc.h>  #include <sys/shm.h>  #include <stdlib.h>  #include "header.h"  *int* main()  {  *int* identifier;  *SharedData* \*control;      if((identifier = shmget(KEY, STRUCTSIZE, 0)) < 0) *// get the identifier of the space*      {          printf("Could not find memory segment.\n");          exit(-1);      }      control = (*SharedData*\*)shmat(identifier, 0, 0); *// attach the sheared space*      if(control <= (*SharedData*\*)(0))      {          printf("Could not attach to shard memory.\n");          exit(-1);      }      control->flag = 2;      while(control->flag != -1) *// do math*      {          if(control->flag == 1)          {              control->sum = control->value1 + control->value2;              printf("%f + %f = %f\n", control->value1, control->value2, control->sum);              control->flag = 0;              printf("Waiting for process 1.\n");          }          else if(control->flag == 2)          {              printf("Waiting for process 1.\n");              control->flag = 0;          }      }      printf("Exiting...\n");      shmdt(control);  } |

**Conclusion:**

This lab gave us experience with using shared memory to allow two unrelated process to communicate with each other. This is sort of like piping, but the processes must be related to each other. Shared memory segments allow for more functionality at the cost of more complicated code.