EECE.4810/EECE.5730: Operating Systems

Spring 2020

Key Questions
Interprocess Communication (IPC) (Lectures 4 & 5)

QUESTIONS

1. What are the two models of interprocess communication? What are the benefits of each?
2. Describe the basics of a producer-consumer problem.
3. Describe the basics of shared memory IPC, using the POSIX shared memory producer/ consumer example programs in the additional handout provided. Be sure to describe (a) how a shared memory segment is established and sized appropriately, (b) how the shared segment is mapped to and removed from a process's address space, (c) how the shared segment can be read or written, and (d) how the shared segment is removed from the file system.
4. Describe message passing through direct communication.
5. Describe message passing through indirect communication.

EXAMPLE

Describe the following pseudo-code, which represents a bounded-buffer implementation of a producer-consumer setup using shared memory IPC.

```
// Basic setup
#define BUFFER_SIZE 10
typedef struct {
     . . .
} item;
item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;


// Producer
item next_produced;
while (true) {
     /* produce an item in next produced */
     while (((in + 1) % BUFFER_SIZE) == out)
          ; /* do nothing */
     buffer[in] = next_produced;
     in = (in + 1) % BUFFER_SIZE;
}

// Consumer
item next_consumed;
while (true) {
     while (in == out)
          ; /* do nothing */
     next_consumed = buffer[out];
     out = (out + 1) % BUFFER_SIZE;
     /* consume the item in next consumed */
}
```

```c
/**
 * Simple program demonstrating shared memory in POSIX systems.
 *
 * This is the producer process that writes to the shared memory region.
 *
 * Figure 3.17
 *
 * @author Silberschatz, Galvin, and Gagne
 * Operating System Concepts - Ninth Edition
 * Copyright John Wiley & Sons - 2013
 *
 * modifications by dheller@cse.psu.edu, 31 Jan. 2014
 *
 * Downloaded from: http://www.cse.psu.edu/~deh25/cmpsc473/notes/OSC/Processes/shm.html
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <errno.h>

void display(char *prog, char *bytes, int n);

int main(void)
{
  const char *name = "/shm-example";    // file name
  const int SIZE = 4096;        // file size

  const char *message0 = "Studying ";
  const char *message1 = "Operating Systems ";
  const char *message2 = "Is Fun!";
  const char *msg_end  = "\n";

  int shm_fd;       // file descriptor, from shm_open()
  char *shm_base;   // base address, from mmap()
  char *ptr;        // shm_base is fixed, ptr is movable

  /* create the shared memory segment as if it was a file */
  shm_fd = shm_open(name, O_CREAT | O_RDWR, 0666);
  if (shm_fd == -1) {
    printf("prod: Shared memory failed: %s\n", strerror(errno));
    exit(1);
```

```
49    }
50
51    /* configure the size of the shared memory segment */
52    ftruncate(shm_fd, SIZE);
53
54    /* map the shared memory segment to the address space of the process */
55    shm_base = mmap(0, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
56    if (shm_base == MAP_FAILED) {
57      printf("prod: Map failed: %s\n", strerror(errno));
58      // close and shm_unlink?
59      exit(1);
60    }
61
62    /**
63     * Write to the mapped shared memory region.
64     *
65     * We increment the value of ptr after each write, but we
66     * are ignoring the possibility that sprintf() fails.
67     */
68    display("prod", shm_base, 64);
69    ptr = shm_base;
70    ptr += sprintf(ptr, "%s", message0);
71    ptr += sprintf(ptr, "%s", message1);
72    ptr += sprintf(ptr, "%s", message2);
73    ptr += sprintf(ptr, "%s", msg_end);
74    display("prod", shm_base, 64);
75
76    /* remove the mapped memory segment from the address space of the process */
77    if (munmap(shm_base, SIZE) == -1) {
78      printf("prod: Unmap failed: %s\n", strerror(errno));
79      exit(1);
80    }
81
82    /* close the shared memory segment as if it was a file */
83    if (close(shm_fd) == -1) {
84      printf("prod: Close failed: %s\n", strerror(errno));
85      exit(1);
86    }
87
88    return 0;
89  }
90
91  void display(char *prog, char *bytes, int n)
92  {
93    printf("display: %s\n", prog);
94    for (int i = 0; i < n; i++)
95      { printf("%02x%c", bytes[i], ((i+1)%16) ? ' ' : '\n'); }
96    printf("\n");
97  }
```

```c
/**
 * Simple program demonstrating shared memory in POSIX systems.
 *
 * This is the consumer process
 *
 * Figure 3.18
 *
 * @author Gagne, Galvin, Silberschatz
 * Operating System Concepts - Ninth Edition
 * Copyright John Wiley & Sons - 2013
 *
 * modifications by dheller@cse.psu.edu, 31 Jan. 2014
 *
 * Downloaded from: http://www.cse.psu.edu/~deh25/cmpsc473/notes/OSC/Processes/shm.html
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <errno.h>
#include <string.h>

void display(char *prog, char *bytes, int n);

int main(void)
{
  const char *name = "/shm-example";    // file name
  const int SIZE = 4096;        // file size

  int shm_fd;       // file descriptor, from shm_open()
  char *shm_base;   // base address, from mmap()

  /* open the shared memory segment as if it was a file */
  shm_fd = shm_open(name, O_RDONLY, 0666);
  if (shm_fd == -1) {
    printf("cons: Shared memory failed: %s\n", strerror(errno));
    exit(1);
  }

  /* map the shared memory segment to the address space of the process */
  shm_base = mmap(0, SIZE, PROT_READ, MAP_SHARED, shm_fd, 0);
  if (shm_base == MAP_FAILED) {
    printf("cons: Map failed: %s\n", strerror(errno));
    // close and unlink?
```

```
49      exit(1);
50    }
51
52    /* read from the mapped shared memory segment */
53    display("cons", shm_base, 64);    // first as bytes, then as a string
54    printf("%s", shm_base);
55
56    /* remove the mapped shared memory segment from the address space of the process
         */
57    if (munmap(shm_base, SIZE) == -1) {
58      printf("cons: Unmap failed: %s\n", strerror(errno));
59      exit(1);
60    }
61
62    /* close the shared memory segment as if it was a file */
63    if (close(shm_fd) == -1) {
64      printf("cons: Close failed: %s\n", strerror(errno));
65      exit(1);
66    }
67
68    /* remove the shared memory segment from the file system */
69    if (shm_unlink(name) == -1) {
70      printf("cons: Error removing %s: %s\n", name, strerror(errno));
71      exit(1);
72    }
73
74    return 0;
75  }
76
77  void display(char *prog, char *bytes, int n)
78  {
79    printf("display: %s\n", prog);
80    for (int i = 0; i < n; i++)
81      { printf("%02x%c", bytes[i], ((i+1)%16) ? ' ' : '\n'); }
82    printf("\n");
83  }
84
85
```