Lab 6, Operating Systems

Jay R Bolton

November 14, 2011

1. *Run producer.c and consumer.c* – Done.
2. *Explain how the programs work*

   consumer.c: The consumer() function opens the provided pipe. It then loops 15 times, waiting three seconds, and calling `readmsg` on the pipe. `readmsg` reads from the pipe and tests the size of its response. If it is greater than zero, it prints that it has read a certain number of bytes and otherwise it exits.

   producer.c: The producer() function opens the provided pipe, then loops 15 times, sleeping 1 second and then doing `sendmsg`. `sendmsg` writes soem text to a buffer and then writes that to the pipe. It the write failed, then it exists.
3. Starting without creating/specifying a FIFO won't work as the program assumes it's user-supplied.
4. Running the programs on the VM did not seem to make any difference. I ran them concurrently by using GNU Screen.
5. The "seminar workshop" code works by loading pointers to `producer` and `consumer` into an array of functions. `which` selects which of those two funcs to do next. `handler` controls `which`. The handler uses signals to indicate when it has begun and ended. The main function loops infinitely, calling either the producer or consumer. Handler is called by the producer and consumer when they are complete.
6. I filled in the missing code and it seems to work.
7. Works fine on ada.
8. Works fine on slacker.
9. I had the producer and consumer generate a random number and loop until either the random number or the buffer boundary. The handler chose the producer or consumer at random. I couldn't really figure out exactly what you wanted.

10. Signals do not necessarily maintain their ordering. Many problems, such as the barbershop problem, that require nested semaphores wouldn't work without ordering. Signals cannot be queued.

My modified code:

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

/*
 * Prototypes
 */

void producer();
void consumer();

/*
 * Global values:
 * cnt: buffer pool index
 * buffers: the buffer pool
 * funcPtrs: the type of the pointer to function
 * func_array: the function array
 * which: which function do I want
 */

static int cnt = 0;
char *buffers[10];

typedef void (*funcPtrs)(void);

funcPtrs func_array[2] = { NULL };

int which;


/*
 * You can enter the handler either when consuming or producing.
 * The handler decides when to switch between producing and consuming
 */


void handler(int i) {
```

```
    signal(SIGUSR1, SIG_IGN);
    printf("In handler with cnt = %d which = %d\n", cnt, which);
    int r;
    srandom(time(0));
    r = random() % 2;
    printf("Random number for handler = %d \n", r);
    which = r;

    signal(SIGUSR1, handler);
}


int main() {
    int i;
    which = 0;
    signal(SIGUSR1, handler);
    func_array[0] = &producer;
    func_array[1] = &consumer;

    /* Initialize Buffers */
    for(i = 0; i < 10; i++) buffers[i] = malloc(80);

    while (1) (*func_array[which])();
}



void producer() {
    int i, r;
    printf("Producing with cnt = %d\n", cnt);
    srandom(time(0));
    r = random() % 10;
    printf("Random number for producer = %d \n", r);

    /* Fill buffers[cnt] with a message and getpid() (eg, use sprintf) */
    for(i = 0; i < r && cnt < 9; i++)  {
      sprintf(buffers[++cnt], "You can't put bacon in a pie!\n");
      printf("Produced: %s and Count = %d\n", buffers[cnt], cnt);
    }

    sleep(2);
    kill(getpid(), SIGUSR1);
}


void consumer() {
```

```
    int i, r;
    printf("Consuming with cnt = %d\n", cnt);
    srandom(time(0));
    r = random() % 10;
    printf("Random number for consumer = %d \n", r);

    /* Print the consumer message in buffers[cnt] */
    for(i = 0; i < r && cnt > 0; i++)
      printf("Consumed: %s and Count = %d\n",buffers[--cnt],cnt);

    sleep(1);
    kill(getpid(), SIGUSR1);
}
```