ASSIGNMENT DESCRIPTION:

/*
Make 2 processes, a reading process and a writing process that access a common buffer with a count variable. Use fork() to create the processes. Create a third process that controls the two processes that were created by the fork() call.

Process A writes to a buffer Z that is of length 100 at a rate of 1 sample per second. It writes consecutive numbers into the buffer starting at 100.

Process B reads the buffer Z, 10 samples at a time. It reads when Process A signals it to read using a semaphore to do the signal. Process B reads the samples and adds a user prompted number to the numbers that it reads from the buffer.

The common buffer will be created using shared memory.

Process A, and B run continuously, that is until Process C sends a signal to them and shuts them down. You can set this signal up using a flag in shared memory, or you can use a semaphore.
*/

IMAGES + OUTPUT:

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>     /* exit, EXIT_FAILURE */
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#include <sys/sem.h>

#define SHARE_SIZE 128

int main()
{
    int j;
    char curr;

    /* Shared memory variables. */
    int shmid;
    key_t memKey, semKey;
    char *buffer;
    // char *buffer;

    /* Semaphore variables and data structures. */
    int id;
    union semun {
        int val;
        struct semid_ds *buf;
        unsigned short *array;
    } argument;

```

```
31      struct sembuf operations[1];
32      int retval;
33
34      /* Let user know that the program is alive. */
35      printf("hello tv land! \n");
36      printf("put a number: ");
37      int num = getchar() - (int)(48);
38
39      /* Set up a semphore that can be used to cause the read part of the
40         fork() to wait for the write portion. */
41      argument.val = 0;
42      semKey = 456;
43
44      /* Create semaphore, id is semKey, second arg is number of
45              semaphores in array to create (just 1), third arg is permissions. */
46      id = semget(semKey, 1, 0666|IPC_CREAT);
47
48      if (id < 0) {
49          printf("Error: Could not create semaphore. \n");
50          exit(id);
51      }
52
53      /* Set the initial value of the semaphore to 0. */
54      if (semctl(id, 0, SETVAL, argument) < 0) {
55          printf("Error: Could not set value of semaphore. \n");
56      }
57      else {
58          printf("Semaphore %d initialized. \n", semKey);
59      }
```

```
61      /* Make a memory key for our shared memory buffer. */
62      memKey = 123;
63      if (fork() == 0) {
64          /* Write part of fork(). */
65
66          /* We need to make sure we are attached to our semaphore. */
67          id = semget(semKey, 1, 0666);
68          if (id < 0) {
69              printf("Error: Write fork() cannot access semaphore. \n");
70              exit(id);
71          }
72
73          /* Create a piece of shared memory. */
74          if ((shmid = shmget(memKey, SHARE_SIZE, IPC_CREAT|0666)) == -1) {
75              printf("Error allocating shared memory \n");
76              exit(shmid);
77          }
78
79          /* Attach to the memory. */
80          if ((buffer = shmat(shmid, NULL, 0)) == (char *)-1) {
81              printf("Could not attach to shared memory.\n");
82              exit(-1);
83          }
84
85          curr = 100;
86          buffer[110] = 0;
87          printf("write process\n");
88          fflush(stdout);
89
```

```
90          /* Write to buffer and also keep write pointer updated. */
91          for(j=0;j<27;j++){
92              /* Send signal to start read process. */
93
94              if(j%10 == 9) {
95
96                  printf("%d ", curr);
97                  fflush(stdout);
98                  buffer[j] = curr;
99                  curr++;
100                 // curr = curr + 1;
101                 buffer[110] = j;
102
103                 /* Set up a semaphore wait operation (P) */
104                 operations[0].sem_num = 0; /* First semaphore in the semaphore array. */
105                 operations[0].sem_op = 1; /* Set semaphore operation to send. */
106                 operations[0].sem_flg = 0; /* Wait for signal. */
107
108                 /* Do the operation. */
109                 retval = semop(id, operations, 1);
110                 if (retval == 0) {
111                     printf("\nWrite process waiting signaled Read process. \n");
112                 }
113                 else {
114                     printf("Error: Write process semaphore operation error. \n");
115                     exit(-1);
116                 }
117                 sleep(1);
118             } else {
119
```

```
120             /* Set curr to the current character. */
121             printf("%d ", curr);
122             fflush(stdout);
123 
124             /* Load the shared buffer. */
125             buffer[j] = curr;
126 
127             /* Increment current character. */
128             curr = curr + 1;
129 
130             buffer[110] = j;
131 
132             /* Sleep for 1 second. */
133             sleep(1);
134         }
135     }
136     /* Unattach from shared memory. */
137     shmdt(NULL);
138 } else {
139     /* Read portion of the fork(). */
140     /* We need to make sure we are attached to our semaphore. */
141     id = semget(semKey, 1, 0666);
142     if (id < 0) {
143         printf("Error: Read fork() cannot access semaphore. \n");
144         exit(id);
145     }
146 
```

```
147         /* Set up a semaphore wait operation (P) */
148         operations[0].sem_num = 0; /* First semaphore in the semaphore array. */
149         operations[0].sem_op = -1; /* Set semaphore operation to wait. */
150         operations[0].sem_flg = 0; /* Wait for signal. */
151
152         /* Do the operation. */
153         retval = semop(id, operations, 1);
154         if (retval == 0) {
155             printf("\nRead process recived signal from Write process.");
156         }
157         else {
158             printf("Error: Read process semaphore operation error. \n");
159             exit(-1);
160         }
161
162
163         /* Create a piece of shared memory. */
164         if ((shmid = shmget(memKey, SHARE_SIZE, IPC_CREAT|0666)) == -1) {
165             printf("Error allocating shared memory \n");
166             exit(shmid);
167         }
168
169         /* Attach to the memory. */
170         // if ((buffer = (char(*))shmat(shmid, NULL, 0)) == (char *)-1) {
171         if ((buffer = shmat(shmid, NULL, 0)) == (char *)-1) {
172             printf("Could not attach to shared memory.\n");
173             exit(-1);
174         }
175
```

```
176
177         /* Read the memory loaded by the parent process. */
178         // int x; printf("\n");
179         int x;
180         do {
181
182             if(buffer[110] % 10 == 9) {
183                 printf("\nshared buffer contents: ");
184                 for(j=0;j<=buffer[110];j++) {
185                     x =  buffer[j] + num;
186                     printf("%d ", x);
187                 }
188                 printf("\n");
189             }
190             sleep(1);
191         } while (buffer[110] != 27);
192         /* Unattach from shared memory. */
193         shmdt(NULL);
194     }
195 }
```

```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE  OPEN EDITORS                1: a.out

plutonium@quanta A4 % ./a.out
hello tv land!
put a number: 3
Semaphore 456 initialized.
write process
100 101 102 103 104 105 106 107 108 109
Write process waiting signaled Read process.

Read process recived signal from Write process.
shared buffer contents: 103 104 105 106 107 108 109 110 111 112
110 111 112 113 114 115 116 117 118 119
Write process waiting signaled Read process.

shared buffer contents: 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122
120 121 122 123 124 125 126
```

CODE:

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>     /* exit, EXIT_FAILURE */
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
```

```
#define SHARE_SIZE 128

int main()
{
   int j;
   char curr;

   /* Shared memory variables. */
   int shmid;
   key_t memKey, semKey;
   char *buffer;
   // char *buffer;

   /* Semaphore variables and data structures. */
   int id;
   union semun {
      int val;
      struct semid_ds *buf;
      unsigned short *array;
   } argument;

   struct sembuf operations[1];
   int retval;

   /* Let user know that the program is alive. */
   printf("hello tv land! \n");
   printf("put a number: ");
   int num = getchar() - (int)(48);

   /* Set up a semphore that can be used to cause the read part of the
     fork() to wait for the write portion. */
   argument.val = 0;
   semKey = 456;

   /* Create semaphore, id is semKey, second arg is number of
         semaphores in array to create (just 1), third arg is permissions. */
```

```
id = semget(semKey, 1, 0666|IPC_CREAT);

if (id < 0) {
    printf("Error: Could not create semaphore. \n");
    exit(id);
}

/* Set the initial value of the semaphore to 0. */
if (semctl(id, 0, SETVAL, argument) < 0) {
    printf("Error: Could not set value of semaphore. \n");
}
else {
    printf("Semaphore %d initialized. \n", semKey);
}

/* Make a memory key for our shared memory buffer. */
memKey = 123;
if (fork() == 0) {
    /* Write part of fork(). */

    /* We need to make sure we are attached to our semaphore. */
    id = semget(semKey, 1, 0666);
    if (id < 0) {
        printf("Error: Write fork() cannot access semaphore. \n");
        exit(id);
    }

    /* Create a piece of shared memory. */
    if ((shmid = shmget(memKey, SHARE_SIZE, IPC_CREAT|0666)) == -1) {
        printf("Error allocating shared memory \n");
        exit(shmid);
    }

    /* Attach to the memory. */
    if ((buffer = shmat(shmid, NULL, 0)) == (char *)-1) {
        printf("Could not attach to shared memory.\n");
        exit(-1);
```

```
}

curr = 100;
buffer[110] = 0;
printf("write process\n");
fflush(stdout);

/* Write to buffer and also keep write pointer updated. */
for(j=0;j<27;j++){
   /* Send signal to start read process. */

   if(j%10 == 9) {

      printf("%d ", curr);
      fflush(stdout);
      buffer[j] = curr;
      curr++;
      // curr = curr + 1;
      buffer[110] = j;

      /* Set up a semaphore wait operation (P) */
      operations[0].sem_num = 0; /* First semaphore in the semaphore array. */
      operations[0].sem_op = 1; /* Set semaphore operation to send. */
      operations[0].sem_flg = 0; /* Wait for signal. */

      /* Do the operation. */
      retval = semop(id, operations, 1);
      if (retval == 0) {
         printf("\nWrite process waiting signaled Read process. \n");
      }
      else {
         printf("Error: Write process semaphore operation error. \n");
         exit(-1);
      }
      sleep(1);
   } else {
```

```
            /* Set curr to the current character. */
            printf("%d ", curr);
            fflush(stdout);

            /* Load the shared buffer. */
            buffer[j] = curr;

            /* Increment current character. */
            curr = curr + 1;

            buffer[110] = j;

            /* Sleep for 1 second. */
            sleep(1);
        }
    }
    /* Unattach from shared memory. */
    shmdt(NULL);
} else {
    /* Read portion of the fork(). */
    /* We need to make sure we are attached to our semaphore. */
    id = semget(semKey, 1, 0666);
    if (id < 0) {
        printf("Error: Read fork() cannot access semaphore. \n");
        exit(id);
    }

    /* Set up a semaphore wait operation (P) */
    operations[0].sem_num = 0; /* First semaphore in the semaphore array. */
    operations[0].sem_op = -1; /* Set semaphore operation to wait. */
    operations[0].sem_flg = 0; /* Wait for signal. */

    /* Do the operation. */
    retval = semop(id, operations, 1);
    if (retval == 0) {
        printf("\nRead process recived signal from Write process.");
    }
```

```
else {
    printf("Error: Read process semaphore operation error. \n");
    exit(-1);
}


/* Create a piece of shared memory. */
if ((shmid = shmget(memKey, SHARE_SIZE, IPC_CREAT|0666)) == -1) {
    printf("Error allocating shared memory \n");
    exit(shmid);
}

/* Attach to the memory. */
// if ((buffer = (char(*))shmat(shmid, NULL, 0)) == (char *)-1) {
if ((buffer = shmat(shmid, NULL, 0)) == (char *)-1) {
    printf("Could not attach to shared memory.\n");
    exit(-1);
}



/* Read the memory loaded by the parent process. */
// int x; printf("\n");
int x;
do {

    if(buffer[110] % 10 == 9) {
        printf("\nshared buffer contents: ");
        for(j=0;j<=buffer[110];j++) {
            x =  buffer[j] + num;
            printf("%d ", x);
        }
        printf("\n");
    }
    sleep(1);
} while (buffer[110] != 27);
/* Unattach from shared memory. */
shmdt(NULL);
```

```
    }
}
```