Ted Diepenbrock A4 11-17-2020

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.

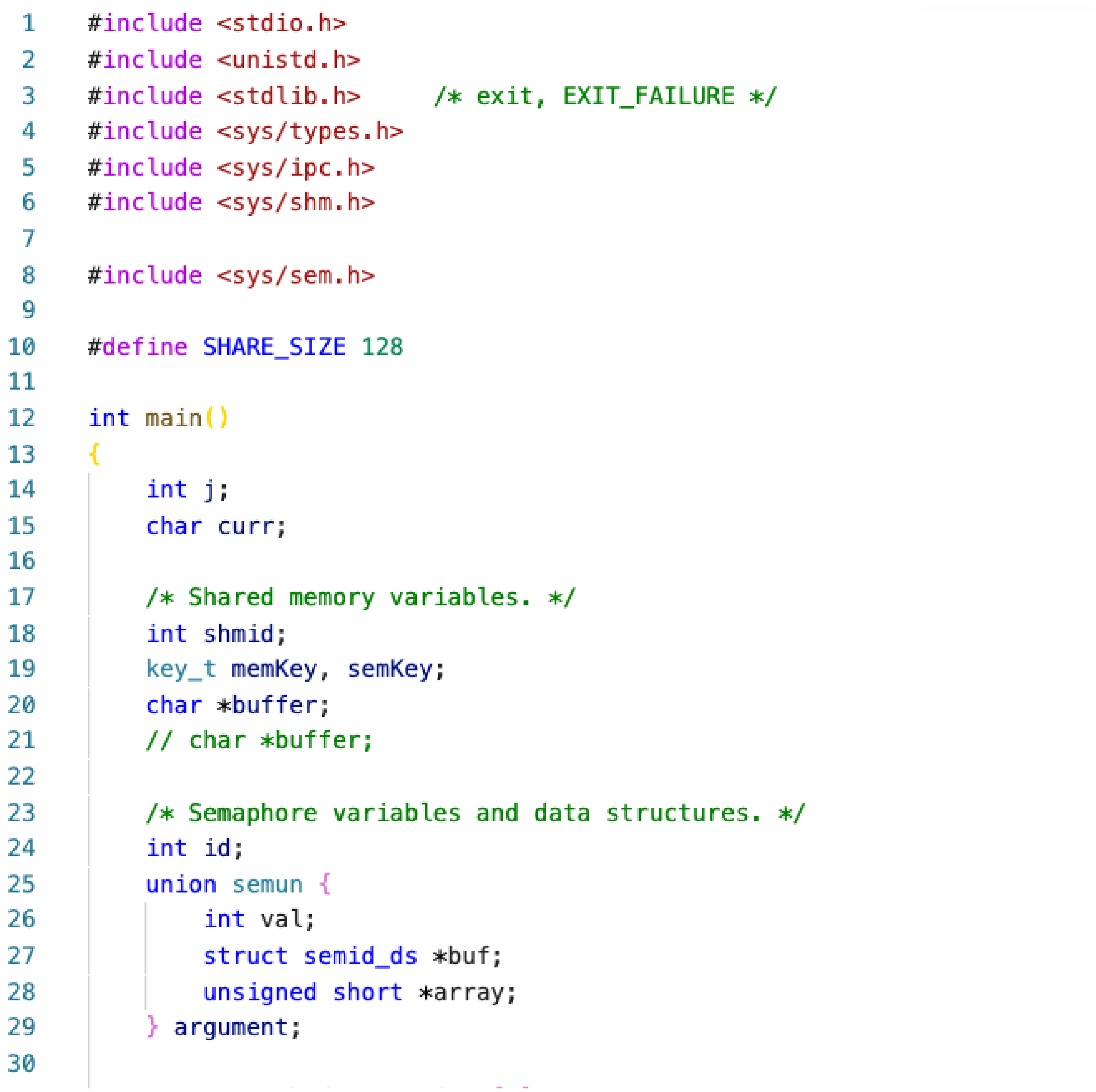
Process A writes to a buffer Z that is of length 100 at a rate of 1 sample per second. It writes consecutive numbers into the buffer starting at 100.

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.

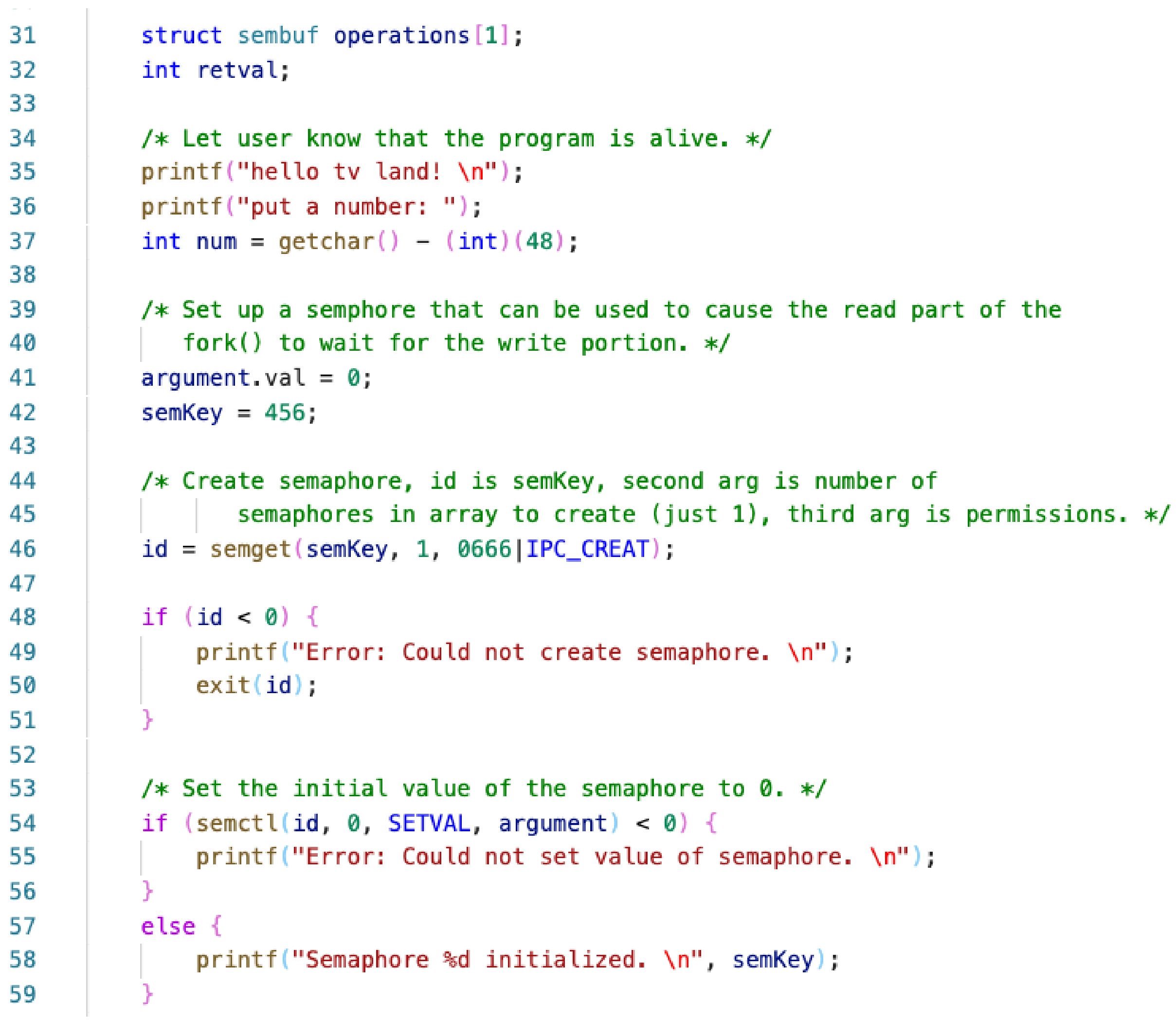
The common buffer will be created using shared memory.

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.
*/

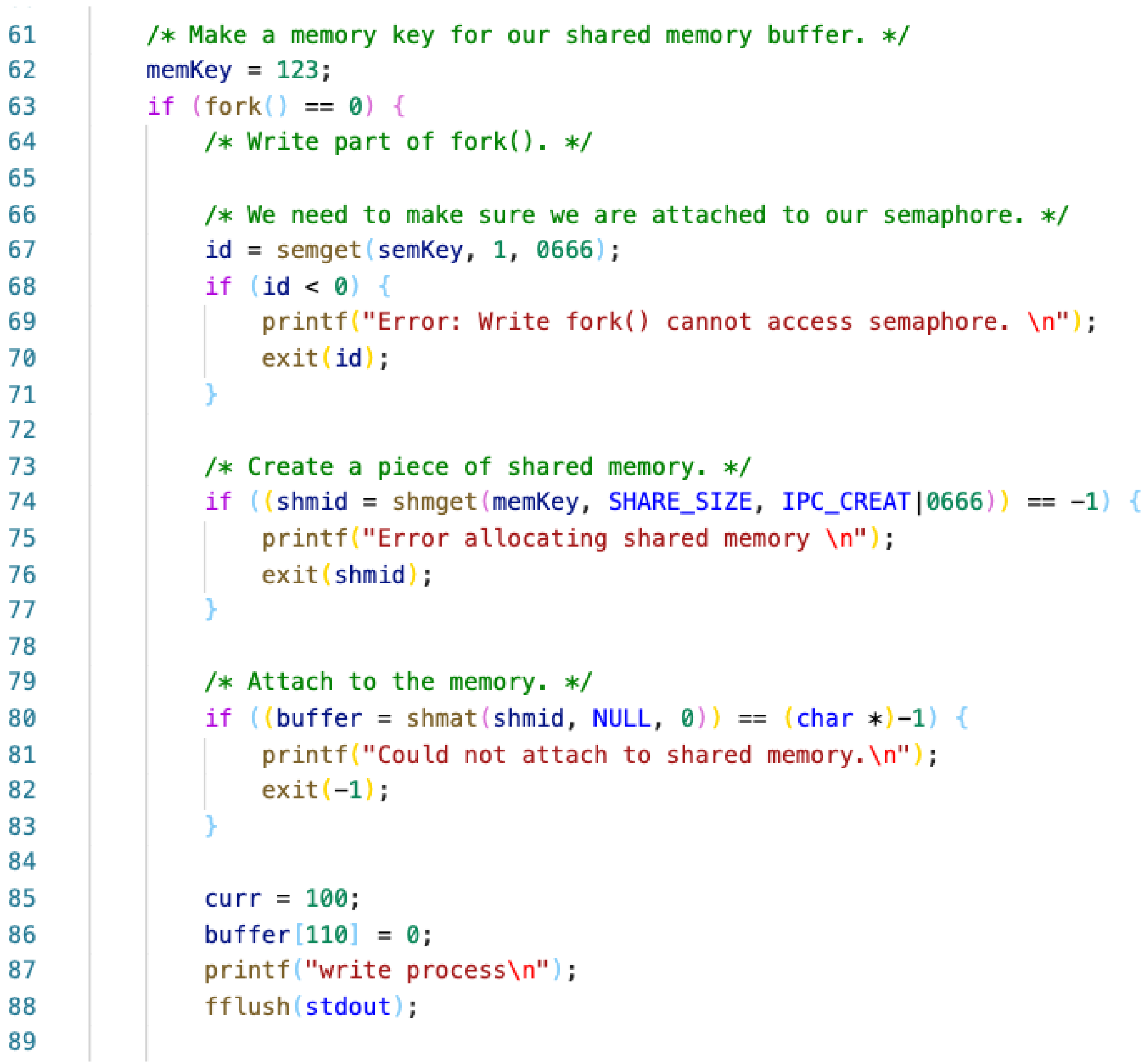
IMAGES + OUTPUT:



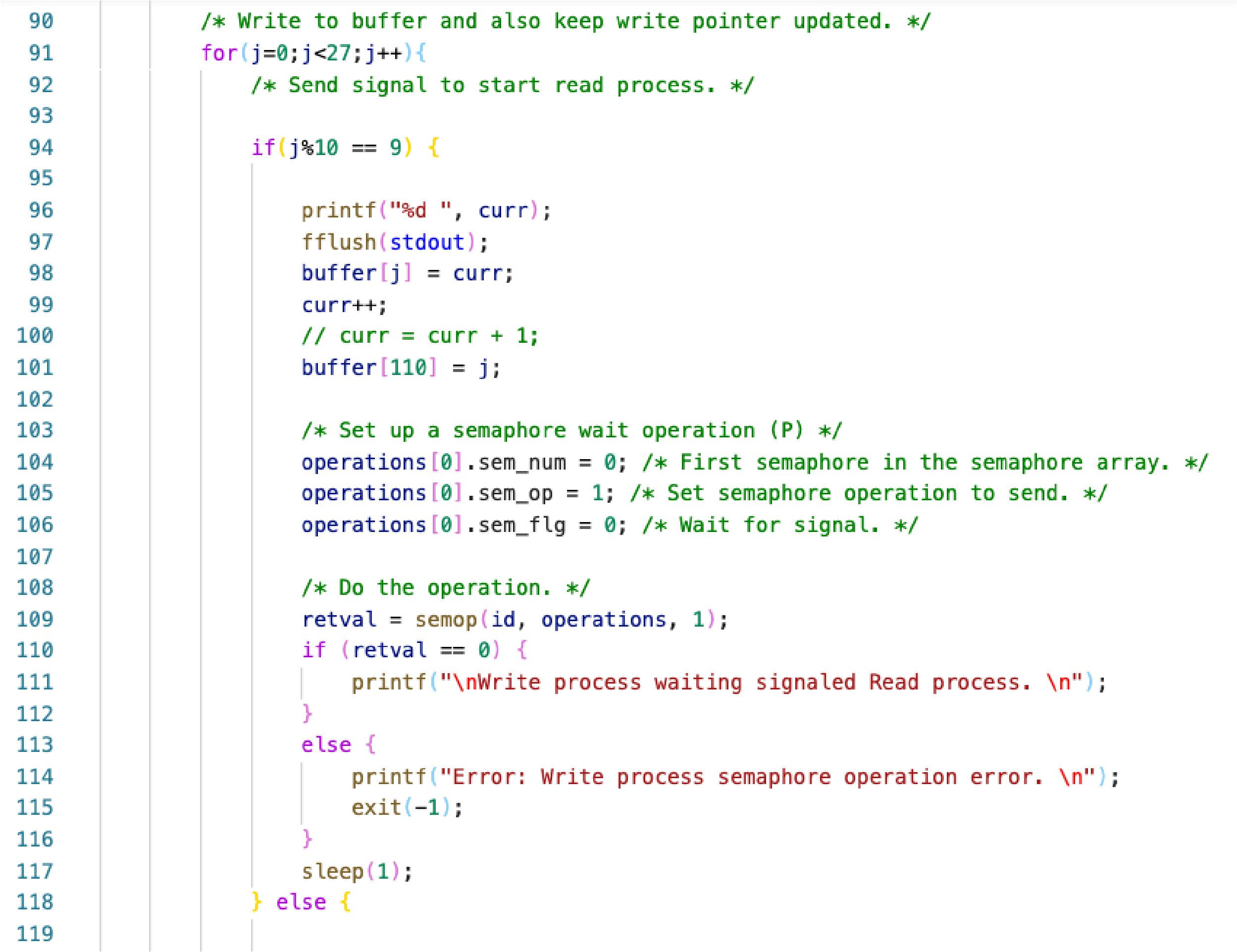
```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>     /* exit, EXIT_FAILURE */
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#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;

```



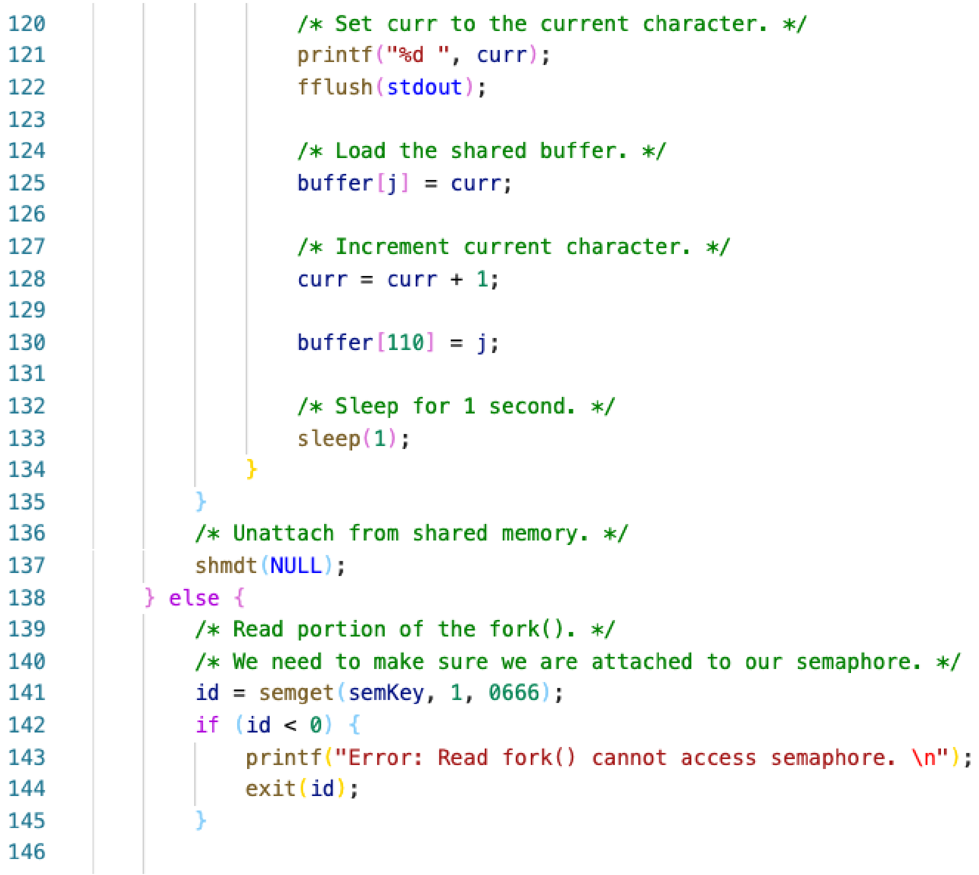
```
    struct sembuf operations[1];
    int retval;

    /* 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);
    }
```



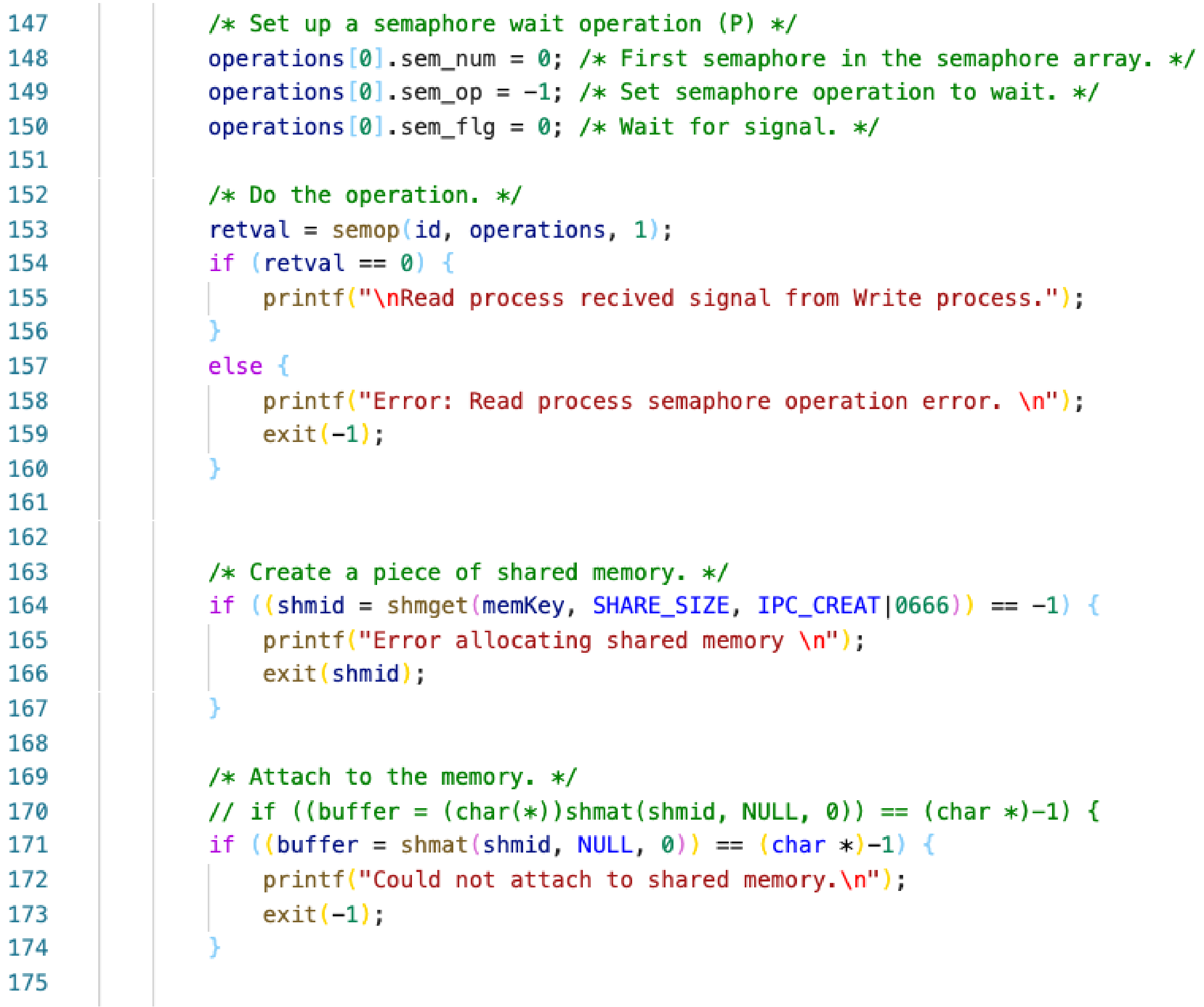
```c
    /* Make a memory key for our shared memory buffer. */
    memKey = 123;
    if (fork() == 0) {
        /* Write part of fork(). */

        /* 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);

```



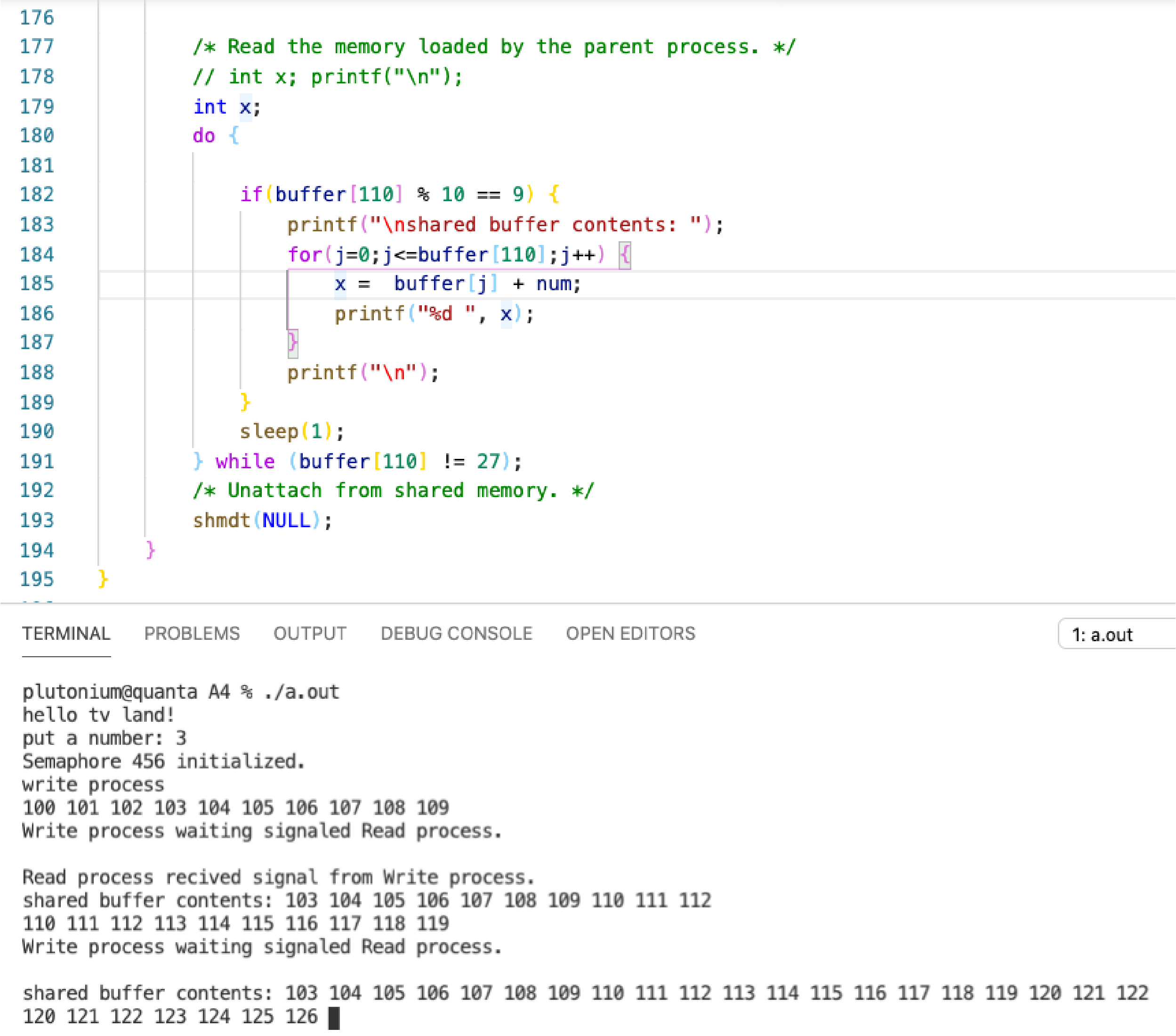
```
        /* Write to buffer and also keep write pointer updated. */
        for(j=0;j<27;j++){
            /* Send signal to start read process. */

            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);
    }
}
```

```
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);
```

```
    }
}
```