IPC

Learning Objectives

Upon completion of this assignment, you should be able:

1. Work in a multiprocess environment
2. Work with IPC, Shared Memory, Semaphores, Message Qs and Signals

New mechanisms you will see and use include:

- system calls for the IPC mechanisms

NAME

IPC: Manage, Compute, Report Suite

SYNOPSIS

```
manage
compute START
report [-k]
```

DESCRIPTION

- manage sets up a shared memory segment and takes update requests
- compute checks for perfect numbers beginning at START
- report prints a summary of the results, and signals a shutdown if -k is given

Exit Status

All programs returns 0 on success and 1 on failure.

DISCUSSION

Part 1: Solution with Process IPC

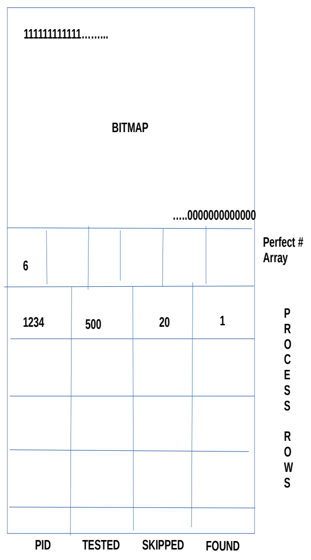
For this assignment you will write 3 related programs to"manage", "report", and "compute" results stored in shared memory.You should hand in files manage.c, report.c, and compute.c defs.h to implement the requirements described below.

Compute's job is to compute perfect numbers. It takes one command line argument, which is the first number to test. It tests all numbers starting at this point, subject to the constraints below. There may be more than one copy of compute running simultaneously.

Manage's job is to create and maintain the IPC elements: a shared memory segment, a message Q, and a semaphore. The shared segment is where the compute processes post their statistics and determine what to test. Manage also keeps track of the active "compute" processes, so that it can signal them to terminate. Manage is ultimately responsible for cleaning up the IPC items it created when it terminates.

Report's job is to read the shared memory segment and report on the perfect numbers found, the total number tested, and for each processes currently computing the number tested, skipped, and found. It should also give a total of these three numbers that includes processes no longer running. If invoked with the "-k" switch, it also is used to inform the Manage process to shut down computation.

The shared memory segment should contain the following data:



- A bit map large enough to contain 2^25 bits. If a bit is off it indicates the corresponding integer has not been tested. The bits are stored as 2^22 bytes.
- An array of integers of length 20 to contain the perfect numbers found.
- An array of "process" structures of length 20, to summarize data on the currently active compute processes. This structure should contain the pid, the number of perfect numbers found, the number of candidates tested, and the number of candidates skipped, since they had already been tested. "Compute" should never test a number already marked in the bitmap.
- You may include a small number of other things in the shared memory segment at your discretion. (eg. Manage's PID so report can signal Manage, a process row accumulating statistics on computes that have terminated, ..).

Compute processes are responsible for updating the bitmap, as well as their own process statistics. However, because of the possible conflicts, "manage" must initialize their process entry for each compute process. You should use a message Q for "compute" to register itself with manage by sending its PID. Manage will select a row and fill in the PID field. This will be a one way communication. Compute blocks on a semaphore after it sends, and compute unlocks the semaphore after it initializes the row. Compute scans the rows for its own PID.

Similarly, "compute" must send "manage" any perfect number found, and manage updates the array of perfect numbers and keeps them in order. There is no need for compute to wait after the send. The message type can distinguish between a PID registrtion and a perfect number report.

Processes that hit the end should wrap around, but stop at their starting point, and exec "report -k". All processes should terminate cleanly on INTR, QUIT, and HANGUP signals. For "compute" processes this means they delete their process entry from the shared memory segment and then terminate. For "manage" it means it sends an INTR signal to all the running computes, sleeps 5 seconds, and then deallocates the shared memory segment, and terminates.

When the -k is flag is used on "report", report sends an INTR to "manage" to force the same shutdown procedure after printing its report.

So that are no conflicts between users, use the last 5 digits of your phone number as a key for the shared memory segment. Also so as not use up unnecessary resources during the debugging phase, use the ps, ipcs, and ipcrm commands to make sure you have not left extraneous processes, shared memory segments, semaphores or message Qs.

SAMPLE OUTPUT:

```
Perfect Number Found: 6 28 496 8128
pid(1203): found: 4, tested: 236136, skipped: 86651
pid(2271): found: 0, tested: 81896,  skipped: 235899
pid(5876): found: 0, tested: 4764,   skipped: 218030
Statistics:
Total found:        4
Total tested:  322996
Total skipped: 540590
```

HANDIN

As usual in */home/*YOURID/cs551/lab4 manage.c compute.c report.c defs.h makefile

Time is short. As a simplification you may store each bit as a 0 or 1 character byte, for a max grade of 90. Limit the number of “bits” to 2^22 (shared memory is limited). If you use this simplification, have manage print out “Storing one bit per byte”.

Part 2: A Thread Solution

In addition to the programs described, you will write a program “tcompute”, which is compatible with ‘manage’ and ‘report’, but is threaded. Tcompute has two types of threads.

- Search Thread : Searches the Bitmap – Single Thread
    - LOOPS
        - Finds next untested number
        - Updates “tested, skipped” in shared mem
        - Puts it in a mutex protected global N
            - signals with condition var 1 that N is available
            - blocks on condition var 2 until N is taken
- Compute Threads : Tests for perfect – 5 Threads
    - LOOPS
        - blocks on condition var 1 until N is available
        - signals with condition var 2 that it took N
        - tests N, if perfect
            - sends message to manage
            - updates “found” in shared memory (another mutex)

Note: Must account for possibility that multiple threads wakeup for same N