Linux Programming Practice Lab

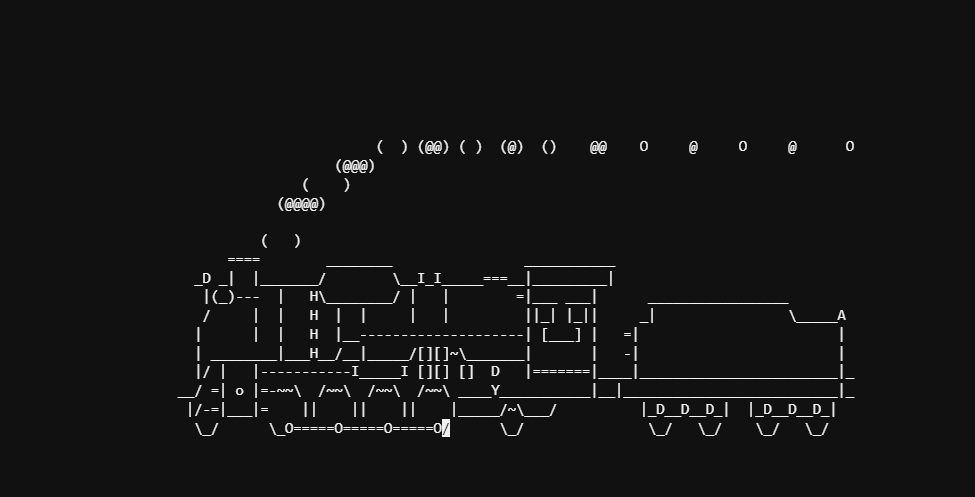
Edith Susanna Andrews

17MIS1048

1. **Script 1**

apt-get install sl

sl

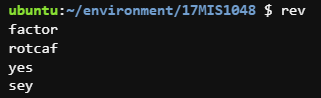


1. **Script 2**

rev

factor

yes

****

1. **Write a C program to implement Simple reader – writer algorithm using shared memory segment with semaphore**

/\* A readers/writers program using a shared buffer and semaphores \*/

#include <sys/types.h>

#include <unistd.h>

#include <stdio.h>

#include <sys/mman.h>

#include <sys/sem.h>

#include <stdlib.h>

#define BUF\_SIZE 5 /\* logical size of buffer \*/

#define SHARED\_MEM\_SIZE (BUF\_SIZE+2)\*sizeof(int) /\* size of shared memory \*/

#define run\_length 10 /\* number of iterations in test run \*/

#define buf\_used 0 /\* semaphore array index to check buffer elts used \*/

#define buf\_space 1 /\* semaphore array index to check buffer elts empty \*/

int sem\_init(void)

{ /\* procedure to create and initialize semaphores and return semaphore id,

assuming two semaphores defined in the given array of semaphores \*/

int semid;

/\* create new semaphore set of 2 semaphores \*/

if ((semid = semget (IPC\_PRIVATE, 2, IPC\_CREAT | 0600)) < 0)

{ perror ("error in creating semaphore");/\* 0600 = read/alter by user \*/

exit (1);

}

/\* initialization of semaphores \*/

/\* BUF\_SIZE free spaces in empty buffer \*/

if (semctl (semid, buf\_space, SETVAL, BUF\_SIZE) < 0)

{ perror ("error in initializing first semaphore");

exit (1);

}

/\* 0 items in empty buffer \*/

if (semctl (semid, buf\_used, SETVAL, 0) < 0)

{ perror ("error in initializing second semaphore");

exit (1);

}

return semid;

}

void P(int semid, int index)

{/\* procedure to perform a P or wait operation on a semaphore of given index \*/

struct sembuf sops[1]; /\* only one semaphore operation to be executed \*/

sops[0].sem\_num = index;/\* define operation on semaphore with given index \*/

sops[0].sem\_op = -1; /\* subtract 1 to value for P operation \*/

sops[0].sem\_flg = 0; /\* type "man semop" in shell window for details \*/

if (semop (semid, sops, 1) == -1)

{ perror ("error in semaphore operation");

exit (1);

}

}

void V(int semid, int index)

{/\* procedure to perform a V or signal operation on semaphore of given index \*/

struct sembuf sops[1]; /\* define operation on semaphore with given index \*/

sops[0].sem\_num = index;/\* define operation on semaphore with given index \*/

sops[0].sem\_op = 1; /\* add 1 to value for V operation \*/

sops[0].sem\_flg = 0; /\* type "man semop" in shell window for details \*/

if (semop (semid, sops, 1) == -1)

{ perror ("error in semaphore operation");

exit (1);

}

}

int main (void)

{ pid\_t pid; /\* variable to record process id of child \*/

/\* shared memory elements \*/

caddr\_t shared\_memory; /\* shared memory base address \*/

int \*in; /\* pointer to logical 'in' address for writer \*/

int \*out; /\* pointer to logical 'out' address for reader \*/

int \*buffer; /\* logical base address for buffer \*/

/\* semaphore elements \*/

int semid; /\* identifier for a semaphore set \*/

/\* local variables \*/

int i\_child, j\_child; /\* index variables \*/

int value; /\* value read by child \*/

/\* set up shared memory segment \*/

shared\_memory=mmap(0, SHARED\_MEM\_SIZE, PROT\_READ | PROT\_WRITE,

MAP\_ANONYMOUS | MAP\_SHARED, -1, 0);

if (shared\_memory == (caddr\_t) -1)

{ perror ("error in mmap while allocating shared memory\n");

exit (1);

}

/\* set up pointers to appropriate places in shared memory segment \*/

buffer = (int\*) shared\_memory; /\* logical buffer starts at shared segment \*/

in = (int\*) shared\_memory + BUF\_SIZE\*sizeof(int);

out = (int\*) shared\_memory + (BUF\_SIZE+1)\*sizeof(int);

\*in = \*out = 0; /\* initial starting points \*/

/\* create and initialize semaphore \*/

semid = sem\_init();

if (-1 == (pid = fork())) /\* check for error in spawning child process \*/

{ perror ("error in fork");

exit (1);

}

if (0 == pid)

{ /\* processing for child == reader \*/

printf ("The reader process begins.\n");

for (i\_child = 0; i\_child < run\_length; i\_child++)

{ P(semid, buf\_used); /\* wait semaphore for something used \*/

value = buffer[\*out];

\*out = (\*out + 1) % BUF\_SIZE;

printf ("Reader's report: item %2d == %2d\n", i\_child, value);

V(semid, buf\_space); /\* signal semaphore for space available \*/

if ((i\_child % 3) == 1)

sleep(1); /\* take time to process every third element \*/

}

printf ("Reader done.\n");

}

else

{ /\* processing for parent == writer \*/

printf ("The writer process begins.\n");

for (j\_child = 0; j\_child < run\_length; j\_child++)

{ P(semid, buf\_space);/\* wait semaphore for space available \*/

buffer[\*in] = j\_child\*j\_child; /\* put data in buffer \*/

\*in = (\*in + 1) % BUF\_SIZE;

printf ("Writer's report: item %2d put in buffer\n", j\_child);

V(semid, buf\_used); /\* signal semaphore for something used \*/

if ((j\_child % 4) == 0)

sleep(1); /\* take time to generate every fourth element \*/

}

wait (pid);

printf ("Writer done.\n");

/\* Remove the semaphore from the system and destroy the set of

semaphores and data structure associated with it. \*/

if (semctl (semid, 0, IPC\_RMID) < 0)

{ perror ("error in removing semaphore from the system");

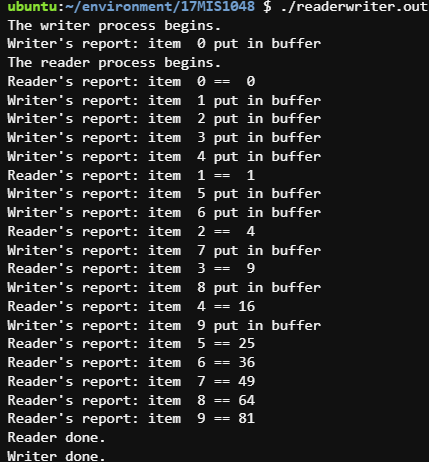
exit (1);

}

}

exit (0);

}

****

1. **Write a bash shell script to monitor the health of your system. Let the details be stored and archived in any folder of your choice.**

#!/bin/bash

echo -e "

Hostname : `hostname`

Kernel Version : `uname -r`

Uptime : `uptime | sed 's/.\*up \([^,]\*\), .\*/\1/'`

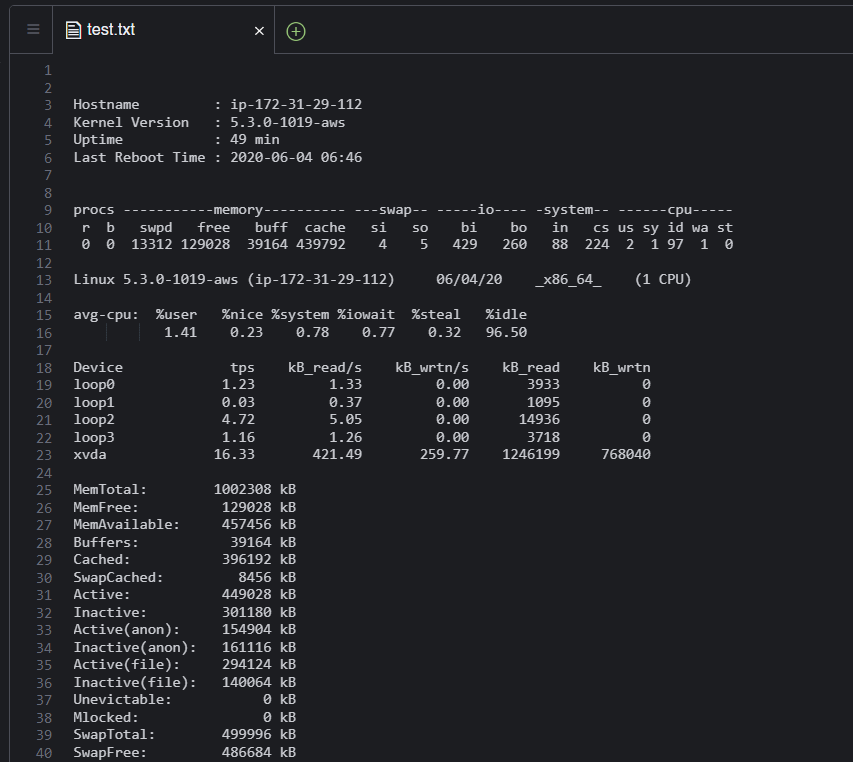
Last Reboot Time : `who -b | awk '{print $3,$4}'`

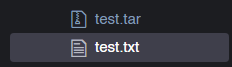
`vmstat`

`iostat`

`cat /proc/meminfo`

"

****

** **