Operating Systems Design 19CS2106A

Mode of submission: Post Handwritten scanned documents in LMS Submission date: on or before 10th November 2020

1. Write a UNIX system program to Solve Producer Consumer problem using POSIX semaphores.

Three conditions must be maintained by the code when the shared buffer is considered as a circular buffer:

- 1. The consumer cannot try to remove an item from the buffer when the buffer is empty.
- 2. The producer cannot try to place an item into the buffer when the buffer is full.
- 3. Shared variables may describe the current state of the buffer (indexes, counts, linked list pointers, etc.), so all buffer manipulations by the producer and consumer must be protected to avoid any race conditions.

Your solution using semaphores should demonstrates three different types of semaphores:

- 1. A binary semaphore named mutex protects the critical regions: inserting a data item into the buffer (for the producer) and removing a data item from the buffer (for the consumer). A binary semaphore that is used as a mutex is initialized to 1. (Obviously we could use a real mutex for this, instead of a binary semaphore.)
- 2. A counting semaphore named nempty counts the number of empty slots in the buffer. This semaphore is initialized to the number of slots in the buffer (NBUFF).
- 3. A counting semaphore named nstored counts the number of filled slots in the buffer. This semaphore is initialized to 0, since the buffer is initially empty

```
OSD
Home Assignment - 4
Home Assignment - 4
Sixrishna, vamsi
vamsi
ite UNIX System program to sol
```

```
1. Aim: write UNIX system program to solve
    producer consumer using posix semaphores
   41 include c stdio. h >
  # include < semaphore. h >
  # include < sys/typesh >
  # include < fontlin>
  # define BUFFER_SIZE
  # define consumer_SLEEP_SFC 3
  # define PRODUCER_SCEEP_SFC 1
  # define KEY 1010
  typedef struct
    int buff [BUFFER_SIZE]
      sent mulex, empty, full;
  3 MEM;
 MEM + menoly ( )
     key-t key = KEY;
     int shmid;
 3
 void init()
    MEM & M = memory();
    seminit (&M -> mutex,1,1);
    sem-init (&M -> empty, 1, BUFFER SIZE);
     sem-init (&M -> tull, 1,0);
```

z

```
producer. c
#include " problem.h"
 void producer ()
 {
      int io,n;
      MEM *s - memory ();
      while (1)
      {
          i++;
          sem_wait ( &s → empty );
          sem-getvalue (25 -> full, 8n);
         ( ) -> buff ) (n) = 1;
          printf ("[PRODUCER] placed item[1/d] \n.i):
      3
   3
   main ()
  init();
     producer ();
   }
consumer . c :
   # include "problem.h"
   void consumer ().
   ¿ mt n;
     MEM AS - memory ())
     while (1)
        sem - wait ( es -> full);
        sen - wait (&s -> mutex);
        sem-post (&s -> mutex);
        sen post (e1 -) empty);
        Sleep ( CONSUMER SLEEP_SFC);
```

2. Considering a system with five processes P0 through P4 and three resources types A, B, C. Resource type A has 10 instances, B has 5 instances and type C has 7 instances. Suppose at time t0 following snapshot of the system has been taken:

Process	Allocation	Max	Available		
	АВС	A B C	АВС		
Po	0 1 0	7 5 3	3 3 2		
P ₁	2 0 0	3 2 2			
P ₂	3 0 2	9 0 2			
P ₃	2 1 1	2 2 2	1		
P ₄	0 0 2	4 3 3			

i. What will be the content of the Need matrix?

	N	Need			
	A	В	C		
Po	7	4	3		
P,	1	2	2		
	6	0	0		
P ₂ P ₃	0	1	1		
Py	4	3	1		

Is the system in safe state? If Yes, then what is the safe sequence? ii.

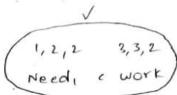
But need < work

for 1=0, Needo = 7,4,3 finish [0] is false

7,4,3 3,4,5 Needo < work

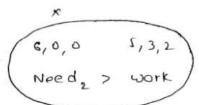
so, Po must be keep in safe sequence

For i=1 Need, = 1,2,2 1,2,2 3,3,2 finish [1] is false. (Need, a work

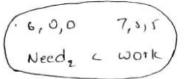


so, P, must be keep in safe sequence

For i= 2 Necd = 6,0,0 finish [2] is take



80, P, must wait For j = 2 Need = 6,0,0 (6,0,0 7,5,5 tinish (2) 13 false



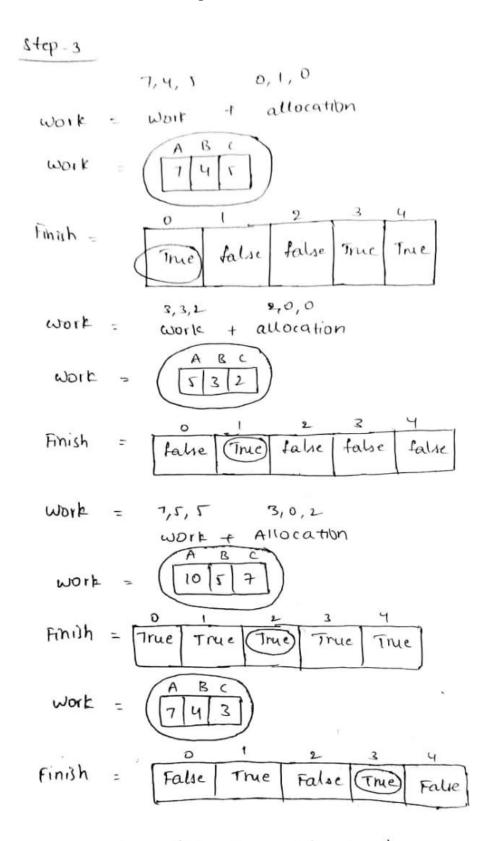
80, P, must be in safe sequence

For i=3 Need = 0,1,1 Finish (3) is false

80, P3 must be in safe sequence

For i= 4 Need 4 = 4,3,1 finish [4] = false

so, Py must be in safe sequence



sty-y: finish[i] = true toi 0 \(\int i\) n

thence, the system is m safe state

CS Scanned with CamScannithe safe step is P1, P3, P4, P0, P2

iii. What will happen if process P₁ requests one additional instance of resource type A and two instances of resource type C?

Рюсем	All	ocat	ion	2	eec	1	Av	aila	ible
	Α	В	c	Α	ß	С	Α	B	<u></u>
Po	0	ı	0	7	4	3	2	3	O
Ρ,	3	0	2	٥	2	0			
P	3	O	2	6	0	0			
3	2	1	ı	0	ţ	1			
P4	0	0	2	1 4	3	ι		KS	

By applying safety algorithm we get to know that the new system is safe so we can grant the reg for process Pi

3. Write a Program using pthreads to demonstrate deadlock.

```
Hindude a pthread ho
3.
      # include < Mdio. h >
     # include a Hellib h>
      pthread_mutex_t resource1, resource2;
      int test = 0;
      void * proct()
         printle ("This is proc! using ");
         pthread_ mutex_lock ( resource 1 );
         usleep (200);
         printf ("In PI trying to get is 2.);
         pthread. muter_lock ( resource 2);
         tut tt)
         print+ (" In proc", got rs2!!);
         pthread - mutex unlock ( resource 2);
         pthread. muter. unlock (resource 1);
        return 0;
      void * proc2()
        pthread_muter_lock (resource 2);
         unsleep (200);
        printf ("In P1 trying to get 2");
          test - -!
       printd (" in proce trying to get rs!!!).
       ncturn 0;
    3
```

```
int main ()

E

pthread - t t1, t2;

pthread - mutex - in it (cresource 1, NULL);

pthread - mutex - init (cresource 2, NULL);

pthread - create (21, NULL, proc1, NULL);

pthread - create (22, NULL, proc2, NULL);

pthread - join (t1, NULL);

pthread - join (t2, NULL);
```

4. Solve Readers-Writers Problem using counter and 2 semaphores.

```
y. -> No reader will be kept waiting unless a writer has the object.

-> writing B performal as Adon as possible semaphore mutex = 1;

semaphore db = 1;

Peader()

{ while (true) {

down (amutex)

reader_count += 1;

if (reader_count == 1)

down (adb);

up (a mutex);

reades_count == 1;
```