OPERATING SYSTEMS

Module3_Part5

Textbook : Operating Systems Concepts by Silberschatz

Producer consumer problem using semaphores

- we present a general structure of this scheme without committing ourselves to any particular implementation;
- We assume that the pool consists of *n* buffers, each capable of holding one item. The mutex semaphore provides mutual exclusion for accesses to the buffer pool and is initialized to the value 1.

  The empty and full semaphores count the number of empty and full buffers. The semaphore empty is initialized to the value n; the semaphore full is initialized to the value 0.

Producer consumer problem using semaphores

```
do {
      . . .
   // produce an item in nextp
      . . .
   wait(empty);
   wait(mutex);
      . . .
   // add nextp to buffer
      . . .
   signal(mutex);
   signal(full);
} while (TRUE);
```

Producer consumer problem using semaphores

```
do {
   wait(full);
   wait(mutex);
      . . .
   // remove an item from buffer to nextc
      . . .
   signal(mutex);
   signal(empty);
      . . .
   // consume the item in nextc
      . . .
} while (TRUE);
```

We can interpret this code as the producer producing full buffers for the consumer or as the consumer producing empty buffers for the producer.

The Readers-Writers Problem

- Suppose that a database is to be shared among several concurrent processes.
- Some of these processes may want only to read the database, whereas others may want to update (that is, to read and write) the database.
- the former is called readers and to the latter called writers.
- if two readers access the shared data simultaneously, no adverse effects will result.
- However, if a writer and some other process (either a reader or a writer) access the database simultaneously, chaos may ensue.
- To ensure that these difficulties do not arise, we require that the writers have exclusive access to the shared database while writing to the database.
- no reader be kept waiting unless a writer has already obtained permission to use the shared object.

Solution to readers writers problem

- In the solution to the first readers-writers problem, the reader processes share the following data structures:

  semaphore mutex, wrt;

  int readcount;

- The semaphores mutex and wrt are initialized to 1; readcount is initializedto 0.
- The semaphore wrt is common to both reader and writer processes.
- The mutex semaphore is used to ensure mutual exclusion when the variable readcount is updated.
- The readcount variable keeps track of how many processes are currently reading the object.
- The semaphore wrt functions as a mutual-exclusion semaphore for the writers.

The Readers-Writers Problem solution

typedef int
semaphore;
semaphore mutex=1;
semaphore wrt=1;
int readcount=0;

```
do {
   wait(wrt);
      . . .
   // writing is performed
      . . .
   signal(wrt);
} while (TRUE);
```

Structure of a

Solution--The Readers-Writers Problem

```
do {
   wait(mutex);
   readcount++;
   if (readcount == 1)
      wait(wrt);
   signal(mutex);
      . . .
   // reading is performed
      . . .
   wait(mutex);
   readcount--;
   if (readcount == 0)
      signal(wrt);
   signal(mutex);
} while (TRUE);
```

Structure of a reader

//gain access to readcount
//increment readcount
//if this is the first process to read db
//prevent writer process to access db
//allow other process to access readcount

//gain access to readcount
//decrement readcounter
//this is the last process to read db
//leave control of db,allow writer process
//allow other process to access readcount