Sep 25, 2006
Lecture 6: Threads

September 5, 2007

1 ADMIN

Today, we will begin discussion of threads (Chapter 5).

2 Review

1. Q: What is busy waiting?

   A: It is the method of waiting for a resource whereby the process repeatedly tests for a variable (semaphore) to change value.

2. Q: Name a synchronization problem that is a generalization of MUTEX.

   A: Producer Consumer Problem.

3. Q: What do we mean by an atomic action? What is Test-and-Set (TAS) or Get-and-Set (GAS)? How do you use them?

   A: Atomic actions are procedures that cannot be interrupted.

   TAS and GAS are atomic actions associated with a semaphore S. The Textbook uses GAS. I prefer to focus on the simpler TAS, where S.TAS() is defined as S.GAS(true).

   S.GAS(val) sets the value of S to val, and returns the OLD value of S.

   We can use it to implement mutual exclusion (for example): before the Critical Section, we execute

   while (S.TAS());

   and after the Critical Section, we execute S=false. It is important to assume that S is false initially.

4. Q: Suppose you have a make file that must be used in CYGWIN as well as in a LINUX environment. E.g., you want execute a file that is named "a.out" in LINUX but named "a.exe" in CYGWIN. How can you make such switch of context convenient?

   A: You can define a "platform" variable called "pf" in the Makefile whose value is "linux" or "cyg". You then test the value of "pf" for each target which depends on the environment:

   ```
   ifndef(pf)
   pf=linux
   pf=cyg
   endif

   ifeq(${pf},linux)
   exe=
   else ifeq(${pf},cyg)
   exe=.exe
   ```

```
endif
endif

target: hw1.c
gcc hw1.c
a${exe}
```

HINT: when you execute the makefile, the current environment of CYGWIN or LINUX is known to the make program. Hence, "pf" have been initialized by your .bashrc program, you need not even need to initialize its value.

3 Wrap Up on Synchronization

- Please read the text on the other toy problems: Readers-Writers Problem, Dining Philosophers Problem.
- WHAT IS BUSY WAITING?
- EXAMPLES OF BUSY WAITING:

  Peterson's solution: while (P2wants and turn=2);

  TAS solution: while (S.TAS());

  REMARK: such solutions are called SPIN LOCK solutions.
- WHY IS BUSY WAITING UNDESIRABLE?

  1. Wasted CPU cycles.

  2. Suppose we have a process H and L with high and low priorities.

  If L enters C.S., then H tries to enter and does busywaiting. It is possible that the scheduler will always schedule H before L. Now they are stuck.
- HOW TO AVOID BUSY WAITING?

  Associate each semaphore S with a queue.

  Each process can block on the queue: S.block();

  Each process can unblock others on the queue: S.unblock();

  We now implement S as an integer-valued semaphore (negative values now allowed).

```
P(S):
S--;
If (S<0) S.block();

V(S):
S++;
If (S<=0) S.unblock();
```

4 THREADS

- While Threads is an important topic on its own right, we have a different reason for threads in this book: we want to explore issues of processes by simulate them as threads at the user-level!
- The best way to understand threads is to think of them as mini-processes. A process can have one or more threads.

- Motivation:

  Your program must break up its task into work that can be done by independent concurrent threads.

- EXAMPLE: Web server application.

  Dispatcher thread: receive incoming requests

  Display thread: obvious

  Network thread: retrieving data from network,

  Worker threads: to handle each request...

  CONSIDER THE ALTERNATIVE (nonthreaded solution): need to carry out each request to completion (but we might be waiting on a disk access, wasting time).

- Benefits of Threads:

  1. Responsiveness (no unnecessary waits)
  2. Resource sharing (threads share common data)
  3. Light weight processes (cheaper than processes for scheduling and on system resources)
  4. Ease of writing code
  5. Ease of exploiting multiprocess architectures

- MODELS of threaded programs.

  (1) Manager/worker: manager thread assigns work to other worker threads. Typically, manager handles all inputs and parcels out work to workers. (can have static or dynamic worker pool)

  (2) Pipeline: like an automobile assembly line.

  (3) Peer: the manager thread is like a worker thread after the initial setup.

- Differences and Similarities:

  1. Each thread is an executional entity, like processes. The name "thread of execution" suggests this very well.
  2. As such, thread has a PC, a register set, a stack.
  3. Unlike processes, all threads in the same process shares the same code section, and shares the data section.
  4. Other resources like files, signals, etc, are also shared process-wide.
  5. Processes are kernel entities. Some threads can be user entities.