University ofOttawa
School of Information Technology and Engineering

Course: CSI3310 – Operating Systems Principles
SEMESTER: Winter 2005

Professor: Stefan Dobrev

Date: February 13 2005
Hour: 13:00-15:00 (2 hours)
Room: Montpetit 202

Midterm Exam

The exam consists of three (3) parts:

Part 1	**Multiple Choice**	**15 points**
Part 2	**Short Answers**	**20 points**
Part 3	**Problem Solving**	**20 points**
Total		**50+5 points**

50 points represents 100% for this exam, and constitutes 20% of the total mark. The exam has 55 points, 5 of them are essentially bonus.

Multiple choice (6x2 +3=15 points, mark the correct answer(s)):

1. Interrupt vector

    a. is a vector pointing to the device that raised the interrupt
    b. is a vector containing the parameters of the raised interrupt
    c. is a table containing the addresses of interrupt handling routines
    d. is a table containing all possible interrupts, together with what might have caused them
    e. is a table of interrupted processes that had not been resumed yet

2. One of the following statements is false:

    In a micro-kernel OS:
    a. Most of the OS functionality is provided by OS components that run as separate server processes.
    b. The OS components have the power to directly talk to each other, without involving the microkernel.
    c. There is increased overhead due to additional context and mode switches involved in every system call.
    d. The OS components can be dynamically loaded and restarted.
    e. The memory space of each OS component is protected from other components, so an error in one component does not overwrite memory of a different component.

3. Preemptive scheduler:

    a. Will preemptively ask a process before execution how long does will it run, and then schedules the process with the shortest execution time first.
    b. Will let the process run, as long as it does not make a system call.
    c. When a higher priority process *p* becomes ready, the scheduler will take the CPU from the currently running process and assign it to the process *p*.
    d. Always schedules the I/O bound processes with highest priority, to achieve high I/O utilization and prevent loss of incoming data.
    e. Will schedule a different process if the time slice of the current process expires.

4. One of the following statements is false:

    Shortest Job First scheduler:
    a. Is a form of priority scheduler using implicitly determined priorities.
    b. Computes the length of the next CPU burst using exponential averaging
    c. Computes the length of the next CPU burst using the following formula:

    $$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n$$

    d. Schedules first the job with the smallest estimated length of the next CPU burst.
    e. Schedules first the job which will have the shortest CPU burst.

5. The thread pools are used to
    a. limit the number of threads, in order to not overwhelm the system
    b. conveniently create new threads to service incoming jobs
    c. reduce the overhead of thread creation and termination
    d. a. and c.
    e. all of them

6. Starvation is not a problem in
    a. FCFS scheduler
    b. SJF scheduler (non-preemptive)
    c. SJF scheduler (preemptive)
    d. Round robin with single priority level
    e. Multilevel Feedback Queue with three queue levels, as long as the time slice of the highest level is less then the time slices of the lower levels.

7. For each of the following statements, mark the appropriate answer:
    a. Long term scheduler is responsible for swapping processes in and out
       True [ ] False [ ]

    b. When a thread calls exec(), the new process gets a copy of each thread of the parent process.
       True [ ] False [ ]

    c. Deferred cancellation is used to postpone the cancellation of a process until its parent is ready to accept the exit value.
       True [ ] False [ ]

    d. In symmetric multiprocessing the central CPU symmetrically distributes the workload to other CPU in order to achieve good load balancing.
       True [ ] False [ ]

    e. Interrupt disabling ensures atomicity of executing the critical section even on a multi-CPU machine
       True [ ] False [ ]

Short answer questions (20 points, answer in the provided space)

1. (3 points) Briefly explain what is DMA and how it works. Why is it used?

2. (3 points) Define multitasking and give one reason why it is used.

3. (3 points) Describe what happens when a system call is made. In particular, explain why the calling user program cannot get control of the system after the CPU has switched to privileged/kernel mode.

4. (5 points) Draw the 5 state process diagram and explain when each of the following transitions occur:

   a. From *new* state to *ready* state

   b. From *ready* state to *wait* state

   c. From *running* state to *ready* state

   d. From *waiting* state to *terminated* state

   e. From *running* state to *wait* state

5. (3 points) Explain the advantages and disadvantages of
    a.) many-to-one mapping of user threads to kernel threads
        - Advantages:

        - Disadvantages:

    b.) one-to-one mapping of user threads to kernel threads
        - Advantages:

        - Disadvantages:

6. (3 points) List the requirements a solution to Critical Section Problem must satisfy. Explain each one of them.

Long questions part (20 points)

Note: Partial solutions will get partial marks.

Problem 1 (8 marks): Simulate Priority Round Robin.

Scheduler and system specification:

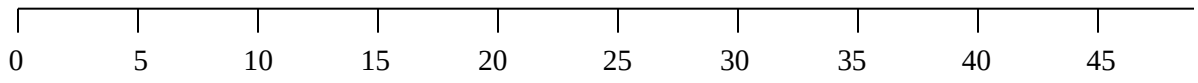
- There are two priority levels: *low* and *high*
- Time quantum for both priority levels is 10 time units
- If a process is preempted because a higher priority process arrives, it remains at the top of its ready queue – it is placed at the end only when its time quantum expires or when its CPU burst is finished.
- Each CPU burst is followed by an I/O operation taking 16 time units
- The I/O operations of different processes overlap and do not interfere, i.e. if P1 starts to wait at time 10 and P2 starts to wait at time 20, P1 will become ready at time 10+16 = 26 and P2 will become ready at time 20+16=36.

Process specification:

- There are four processes P0, P1, P2 and P3
- Processes P0 and P2 arrive at time 0, processes P1 and P3 arrive at time 10
- P0 and P1 are of *high* priority, P2 and P3 are of *low* priority
- The CPU burst time of processes are as follows:
  P0: 2, 3, 2, 3
  P1: 5, 14, 7, 9
  P2: 12, 3, 18, 4
  P3: 15, 25, 32, 2

Your task:

- Simulate the Round Robin scheduler scheduling these 4 processes for the first 40 time steps
- In the following line, mark the times of the context switches and mark which process is executing at a given moment:

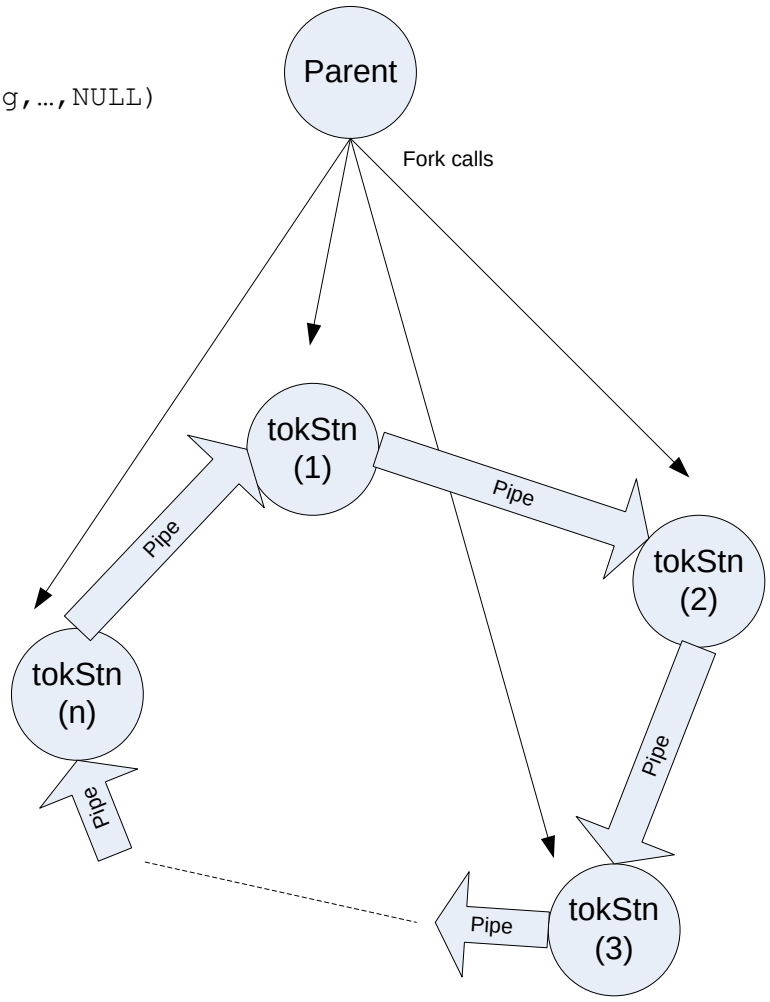


Problem 2 (12 marks): Create nr child processes connected in a ring.

Detailed specification: The function `void createRing(int nr)` is called in a parent process to create `nr` child processes where a pipe connects the standard output of a process to the standard input of another process to form a ring as shown in the diagram below. Complete the function with the appropriate `pipe`, `fork`, `dup2`, and `execlp` calls to create such a ring of child processes. Child processes should only have stdin, stdout, and stderr fd's open when the `execlp()` is called. The child processes should run the `tokStn` program (with no arguments).

Synopsis of system calls available:

```
int fork()
int dup2(int oldfd, int newfd)
int execlp(char *file, char *arg,…,NULL)
int pipe(int filedes[2])
```



```
void createRing(int nr)
{
   int pipefd[2];
   int fdRead,fdWrite;
   int i;


   for(i=0 ; i<nr-1; i++)
   {









   }









      close(fdRead);
      write(fdWrite,"t",1);  /* prime the pump */
      close(fdWrite);
}
```

University ofOttawa
School of Information Technology and Engineering

Course: CSI3310 – Operating Systems Principles
SEMESTER: Winter 2005

Professor: Stefan Dobrev

Date: February 13 2004
Hour: 13:00-15:00 (2 hours)
Room: Montpetit 202

Midterm Exam

The exam consists of three (3) parts:

Part 1	**Multiple Choice**	**15 points**
Part 2	**Short Answers**	**20 points**
Part 3	**Problem Solving**	**20 points**
Total		**50+5 points**

50 points represents 100% for this exam, and constitutes 20% of the total mark. The exam has 55 points, 5 of them are essentially bonus.

Multiple choice (6x2 +3=15 points, mark the correct answer(s)):

1. Interrupt vector

    a. is a vector pointing to the device that raised the interrupt
    b. is a vector containing the parameters of the raised interrupt
    c. is a table containing the addresses of interrupt handling routines
    d. is a table containing all possible interrupts, together with what might have caused them
    e. is a table of interrupted processes that had not been resumed yet

2. One of the following statements is false:

    In a micro-kernel OS:
    a. Most of the OS functionality is provided by OS components that run as separate server processes.
    b. The OS components have the power to directly talk to each other, without involving the microkernel.
    c. There is increased overhead due to additional context and mode switches involved in every system call.
    d. The OS components can be dynamically loaded and restarted.
    e. The memory space of each OS component is protected from other components, so an error in one component does not overwrite memory of a different component.

3. Preemptive scheduler:

    a. Will preemptively ask a process before execution how long does will it run, and then schedules the process with the shortest execution time first.
    b. Will let the process run, as long as it does not make a system call.
    c. When a higher priority process *p* becomes ready, the scheduler will take the CPU from the currently running process and assign it to the process *p*.
    d. Always schedules the I/O bound processes with highest priority, to achieve high I/O utilization and prevent loss of incoming data.
    e. Will schedule a different process if the time slice of the current process expires.

4. One of the following statements is false:

    Shortest Job First scheduler:
    a. Is a form of priority scheduler using implicitly determined priorities.
    b. Computes the length of the next CPU burst using exponential averaging
    c. Computes the length of the next CPU burst using the following formula:

    $$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n$$

    d. Schedules first the job with the smallest estimated length of the next CPU burst.
    e. Schedules first the job which will have the shortest CPU burst.

5. The thread pools are used to
    a. limit the number of threads, in order to not overwhelm the system
    b. conveniently create new threads to service incoming jobs
    c. reduce the overhead of thread creation and termination
    d. a. and c.
    e. all of them

6. Starvation is not a problem in
    a. FCFS scheduler
    b. SJF scheduler (non-preemptive)
    c. SJF scheduler (preemptive)
    d. Round robin with single priority level
    e. Multilevel Feedback Queue with three queue levels, as long as the time slice of the highest level is less then the time slices of the lower levels. *(starves the lowest priority level)*

7. For each of the following statements, mark the appropriate answer:
    a. Long term scheduler is responsible for swapping processes in and out
    True [ ] False [x] *(that is the job of mid-term scheduler)*

    b. When a thread calls exec(), the new process gets a copy of each thread of the parent process.
    True [ ] False [x] *(that might be the case with fork() ,but exec() replaces all threads with a new process, starting with a single thread)*

    c. Deferred cancellation is used to postpone the cancellation of a process until its parent is ready to accept the exit value.
    True [ ] False [x] *(deferred cancellation tells the thread/process to terminate, it is up to the thread to do that, the parent has nothing to do with that)*

    d. In symmetric multiprocessing the central CPU symmetrically distributes the workload to other CPU in order to achieve good load balancing.
    True [ ] False [x] *(In symmetric multiprocessing, there is no central CPU, all CPUs have their own ready queues and use the same algorithm to balance them)*

    e. Interrupt disabling ensures atomicity of executing the critical section even on a multi-CPU machine
    True [x] False [ ] *(Interrupt disabling ensures atomicity, no matter what. What is does not ensure in multi-CPU machine is mutual exclusion)*

Short answer questions (20 points, answer in the provided space)

1. (3 points) Briefly explain what is DMA and how it works. Why is it used?

DMA stands for ***d****irect* ***m****emory* ***a****ccess.* It allows the devices (disk, …) to write/read a block of data directly to/from memory without involving the CPU. The CPU just tells the DMA controller (or device's DMA controller) what it wants to have done, and then receives an interrupt only when the transfer has been finished (or an error occurred). This offloads the interrupt handling and piece-vice data copying from the CPU.

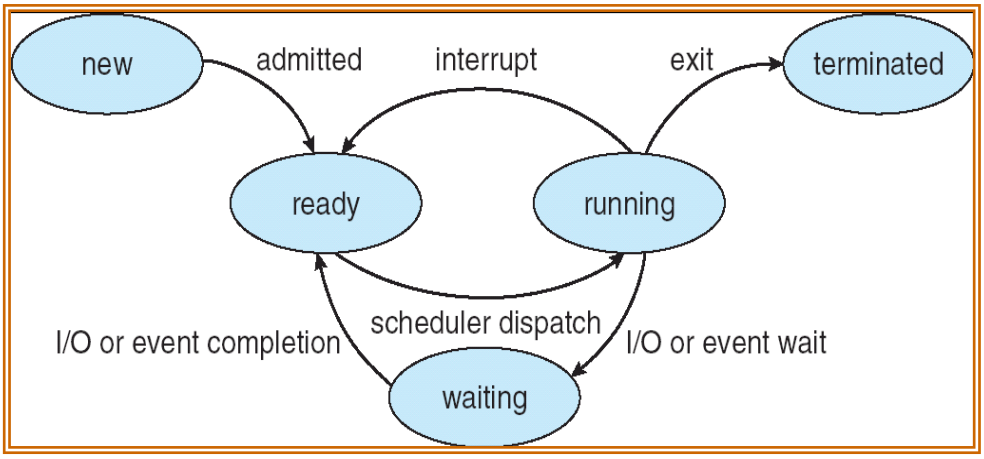
2. (3 points) Define multitasking and give one reason why it is used.

In a multitasking system there are several processes in the memory ready for execution. This allows to schedule another process when the currently running process blocks, achieving higher resource utilization. Moreover, in a multitasking system the processes are scheduled in a time-shared manner (using preemptive scheduler), enabling each user to use the system in an interactive way.

3. (3 points) Describe what happens when a system call is made. In particular, explain why the calling user program cannot get control of the system after the CPU has switched to privileged/kernel mode.

When a system call is made, a software interrupt instruction (called trap) is executed. This switches the CPU to a kernel/privileged mode and the control is given to the location specified in the kernel's trap table containing the addresses of routines for servicing system calls. The calling program can only specify which of the kernel provided routines will be executed, but cannot force the system to execute its (caller's) code.

4. (5 points) Draw the 5 state process diagram and explain when each of the following transitions occur:



a. From *new* state to *ready* state

When a process is admitted to the system by the long-term scheduler

b. From *ready* state to *wait* state

This cannot happen by process's own action. If a suspend signal is sent to the process, it will wait until a resume signal is received.

c. From *running* state to *ready* state

Time quantum expired. Preempted by a higher priority process.

d. From *waiting* state to *terminated* state

Again, cannot directly happen by the action of the process itself. Can happen if somebody sends kill signal to the process, or when the process is caught in cascading termination

e. From *running* state to *wait* state

Called a blocking system call (i.e. blocking I/O operation, wait on a semaphore, wait until child terminates, blocking messaging…)

5. (3 points) Explain the advantages and disadvantages of
    a.) many-to-one mapping of user threads to kernel threads
        - Advantages:

        Efficient thread management operations (creation, synchronization, …), as the work is done in the library, without the overhead of mode switching

        - Disadvantages:

        One thread blocks, all threads block. Cannot be used to distribute the workload to several CPUs.

    b.) one-to-one mapping of user threads to kernel threads
        - Advantages:

        Full flexibility – one thread blocks, others are still scheduler by the scheduler, can spread the work on several CPUs

        - Disadvantages:

        Thread management operations more costly, as the overhead of full system calls is involved.

6. (3 points) List the requirements a solution to Critical Section Problem must satisfy. Explain each one of them.

Mutual exclusion: only one process can be in the critical section at any moment of time
Progress: if some processes want to enter the CS and there is no process in the CS, eventually one of the processes will enter the CS.
Bounded Waiting: (starvation free) a process waiting to enter the CS can be overtaken only finite number of times before being give access to the CS

Long questions part (20 points)

Note: Partial solutions will get partial marks.

Problem 1 (8 marks): Simulate Priority Round Robin.

Scheduler and system specification:

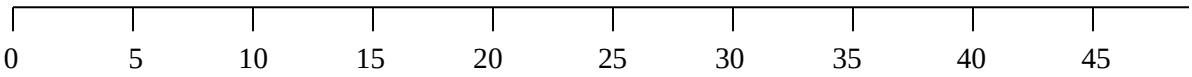
- There are two priority levels: *low* and *high*
- Time quantum for both priority levels is 10 time units
- If a process is preempted because a higher priority process arrives, it remains at the top of its ready queue – it is placed at the end only when its time quantum expires or when its CPU burst is finished.
- Each CPU burst is followed by an I/O operation taking 16 time units
- The I/O operations of different processes overlap and do not interfere, i.e. if P1 starts to wait at time 10 and P2 starts to wait at time 20, P1 will become ready at time 10+16 = 26 and P2 will become ready at time 20+16=36.

Process specification:

- There are four processes P0, P1, P2 and P3
- Processes P0 and P2 arrive at time 0, processes P1 and P3 arrive at time 10
- P0 and P1 are of *high* priority, P2 and P3 are of *low* priority
- The CPU burst time of processes are as follows:
  P0: 2, 3, 2, 3
  P1: 5, 14, 7, 9
  P2: 12, 3, 18, 4
  P3: 15, 25, 32, 2

Your task:

- Simulate the Round Robin scheduler scheduling these 4 processes for the first 40 time steps
- In the following line, mark the times of the context switches and mark which process is executing at a given moment:



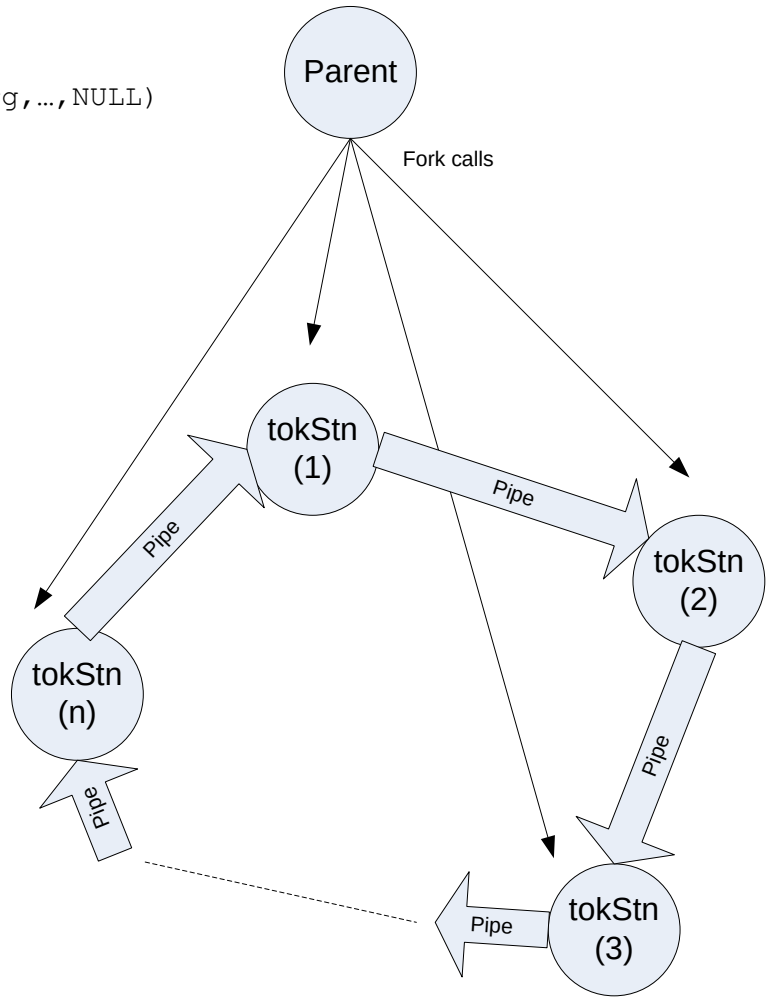
Time	Process	Comment
0	P0	Highest priority scheduled, P2 ready, P1 and P3 not present yet
2	P2	P0 goes to I/O wait, P1 and P3 not present yes
10	P1	P1 and P3 arrive, P2 preempted by P1, P0 still at I/O wait
15	P2	P1 goes to I/O wait, P2 given chance to finish its time slice
17	P3	P2 finished its time slice of 10, the CPU is given to P3 using round robin
18	P0	P0 finishes I/O, becomes ready and preempts P3
21	P3	P0 goes to I/O wait, P3 given chance to finish its time slice
30	P2	P3 finishes its time slice, P0 and P1 still at I/O wait, P2 scheduled
31	P1	P1 finished I/O, becomes ready and preempts P2
41	P0	P1's time slice expired, P0 is scheduled using round robin

Problem 2 (12 marks): Create nr child processes connected in a ring.

Detailed specification: The function `void createRing(int nr)` is called in a parent process to create `nr` child processes where a pipe connects the standard output of a process to the standard input of another process to form a ring as shown in the diagram below. Complete the function with the appropriate `pipe`, `fork`, `dup2`, and `execlp` calls to create such a ring of child processes. Child processes should only have stdin, stdout, and stderr fd's open when the `execlp()` is called. The child processes should run the `tokStn` program (with no arguments).

Synopsis of system calls available:

```
int fork()
int dup2(int oldfd, int newfd)
int execlp(char *file, char *arg,…,NULL)
int pipe(int filedes[2])
```



```
void createRing(int nr) {
    int oriPipe[2];   // the first pipe created
    int childPipe[2]; // the next pipe connecting to the next child
    int fdRead;  // the read end of the last pipe created
    int fdWrite; // the write end of the very first pipe created
    int i;
    char stnnum[20];;

    pipe(oriPipe);
    fdRead = oriPipe[0];
    fdWrite = oriPipe[1];

    // create nr-1 children in a line
    for(i=0 ; i<nr-1 ; i++) {
        sprintf(stnnum,"%d",i);
        pipe(childPipe);
        if (fork()==0)  {  /* In child */
            /* the reads will be from the last pipe created */
            dup2(fdRead, 0);
            // 0 fd will work from now, no need for fdRead
            // fdWrite is inherited but unneeded
            close(fdRead);
            close(fdWrite);

           /* the writes will be into the new pipe */
            dup2(childPipe[1],1);
            // will use only the write end of the childPipe
            // but that is already copied to fd 1
            close(childPipe[1]);
            close(childPipe[0]);

            // execute the tokstn program
            execlp("tokstn","tokstn",stnnum,NULL);
        }
        // the main program after creating this chain of children,
        // needs only the read end of the last pipe
        close(fdRead);  /* no longer needed */
        fdRead = childPipe[0];  /* set to read end of new pipe */
        close(childPipe[1]);   /* do no need this any more */
    }

    // Now create the last child and close the cycle
    /* use pipe saved fds (write end of original pipe */
    /* and read end of last pipe created */
    sprintf(stnnum,"%d",i);
    if (fork()==0)  { /* In child */
        dup2(fdRead,0);  /* read from the last pipe created */
        dup2(fdWrite,1);  /* write to the first pipe created */
        close(fdRead);
        close(fdWrite);
        execlp("tokstn","tokstn",stnnum,NULL);
    }
    close(fdRead);
   /* the main program writes into the cycle, to prime the pump */
    write(fdWrite,"t",1);
    close(fdWrite);
}
```