University of Waterloo
Midterm Examination
Term: Fall Year: 2007

Solution

———begin solution———

```
Grade breakdown:

              Q1      Q2      Q3      Q4      Q5      Q6     Total
MAX           12       8      14      10       9       6        59
AVG         8.99    4.91    7.52    6.98    7.40    2.92     38.71
OUT OF        13       8      14      10       9       6        60
AVG%       69.15   61.40   53.69   69.78   82.17   48.72     64.52

Max Grade = 57/60 = 95%.

10      0.00
20      0.00    # of 10s
30      1.00    # of 20s
40      4.00    # of 30s
50      5.00
60     20.00
70     23.00    # of 60s
80     29.00    # of 70s
90      6.00    # of 80s
100     3.00    # of 90s and 100s

Total 91.00
```

———end solution———

Problem 1 (13 marks)

a. **(4 mark(s))** Explain two different methods that could be used to supply synchronization primitives to user-level threads to allow them to synchronize with each other and discuss the pros and cons of the two approaches.

——————begin solution——————

It turned out that this question was interpretted many different ways by different students. Most missed the key component of the question – the fact that we want to know about user-level threads.

What we were hoping for was: One could build synchronization mechanisms using atomic instructions (like test_and_set or swap).
Pros: no sys calls
Cons: may be inefficient, lock has high contention because there is no way to block someone waiting for a lock.

The operating system could supply system calls for synchronization. This really only works if there are more than on kernel threads in which the user threads can run. Pros: can actually block someone waiting for a lock Cons: may be be more expensive if lock isn't contended.

What we mostly got and mostly accepted.
A comparison of two different synchronization methods. E.g., Semaphores with Locks, Semaphores with CVs+Locks. For these the comparisons needed to be correct and the pros and cons needed to be listed and correct.

Some people compare CVs with Locks. This doesn't make as much sense because you can't have/use CVs without locks.

——————end solution——————

b. **(5 mark(s))** Mark each of the following as either: (I) interrupt, (E) exception, or (N) neither.

(a) Timer

(b) Procedure Call

(c) Segmentation Violation

(d) Divide by Zero

(e) System Call

——————begin solution——————

(a) Timer (I)

(b) Procedure Call (N)

(c) Segmentation Violation (E)

(d) Divide by Zero (E)

(e) System Call (E or I but not N)
The system call will be implemented by generating exception on some processors or by an interrupt on others.

——————end solution——————

c. **(2 mark(s))** What is the difference between deadlock and starvation?

——————begin solution——————

Deadlock means that the processes involved in deadlock are not able to do anything because they each require one or more resources the others are holding.
Starvation means that one ore more processes is unable to gain access to the critical section. In this comparison the difference is that the processes involved in deadlock NEVER run. While in this case the process being starved actually gets a chance to run but it never gets access to the critical section.

Another type of starvation is due to poor/unlucky scheduling. E.g., jobs of higher priority arrive and are run before a job of lower priority has an opportunity to run.

Deadlock does NOT necessarily involve ALL processes. Two processes could be deadlocked and all other processes could continue.

——————end solution——————

d. **(2 mark(s))** A multi-threaded program running on a system with a *single processor* that does not synchronize access to shared variables will result in data corruption or some other incorrect behavior?

- (a) Always
- (b) Sometimes
- (c) Never

Explain your answer:

——————begin solution——————

(b) Sometimes... just because a race condition exists in code does not mean that a particular execution will encounter it.

——————end solution——————

Problem 2 (8 marks)

a. **(2 mark(s))** Under what conditions might two or more processes share common physical frames. Explain how and why this would be done?

————————begin solution————————

If the two processes are executing the same program, they could share the .text and .rodata pages. This is done by mapping virtual pages in both processes to the same physical frames (in the page table/TLB). This could be done to save memory.

————————end solution————————

b. **(2 mark(s))** Name one of the key operating system abstractions that user-level threads from the same process must share and explain why it is shared.

————————begin solution————————

Preferred answer:
Address space because they are executing the same code and they are part of the same process.
or
Open files, again because they are part of the same process.

————————end solution————————

c. **(2 mark(s))** Explain what is meant by the context of a thread?

————————begin solution————————

The context of a thread is all of the information that would be required to keep track of its execution state. This includes:
o the contents of the processors registers including the current instruction being executed (pc/ip) and the stack pointer (sp).
o the stack being used is technically not part of the context (in terms of what needs to be saved and restored during a context switch) but it must be unique for each thread

————————end solution————————

d. **(2 mark(s))** In OS/161 different parts of a thread's context can be saved and restored at different points during the kernel's execution. Describe where these different points occur (logically) and why they are necessary.

————————begin solution————————

The first point is at the time of an exception (this is a mode switch) that occurs when a system call, exception or interrupt occur. This saves the state of the execution of the process in user space so that when execution returns from the kernel that user process can be resumed exactly where it left off.

The second point is at the time of a context switch. Here we are switching from the current thread that is executing in the kernel to a new thread that we want to run (i.e., we are dispatching a new thread). We need to save context here because one thread (the currently running thread) is blocking. Later if/when we want to resume execution of that blocked thread we need a copy of its processor execution state so that it can be resumed.

————————end solution————————

Problem 3 (14 marks)
There have been some accidents recently on the one-lane bridge near Conestogo. To prevent future accidents, traffic lights have been installed at either end of the bridge to synchronize the traffic going in different directions. A car can only cross the bridge if there are no cars going the opposite direction on the bridge. Sensors at either end of the bridge detect when cars arrive and depart from the bridge, and these sensors control the traffic lights. Below is a skeleton implementation of two routines, Arrive() and Depart(). You may assume that each car is represented by a thread, and threads call Arrive() when they arrive at the bridge and Depart() when the leave the bridge. Threads pass their direction of travel as input to the routines.

```
#define DIR_OPEN   (0)                          struct lock *lock = lock_create("Lock");
#define DIR_NORTH  (1)                          struct cv *cv = cv_create("CV");
#define DIR_SOUTH  (2)                          int curdir = DIR_OPEN; int numcars = 0;

    Arrive(int mydir)                                 Depart(int mydir)
    {                                                 {
A.1                                           D.1
A.2   while (curdir != mydir &&               D.2     numcars--;
A.3          curdir != DIR_OPEN)  {           D.3
A.4                                           D.4     if (numcars == 0) {
A.5      ; /* spin doing nothing */           D.5
A.6                                           D.6        dir = DIR_OPEN;
A.7    }                                      D.7
A.8                                           D.8     }
A.9    numcars++;                             D.9
A.10                                          D.10
A.11   curdir = mydir;                        D.11
A.12                                          D.12
     }                                             }
```

a. **(2 mark(s))** The code above doesnt’t properly synchronize access to shared resources. Outline an execution sequence (referring to the numbered statements) where two threads can cause two cars to travel on the bridge in opposite directions at the same time.

————————begin solution————————

Note: many different solutions are possible. One example is:
Thread 1 (North): A.2, curdir = DIR_OPEN, ctxt switch, after waking enters A.9
Thread 2 (South): A.2, curdir = DIR_OPEN, ctxt switch, after waking enters A.9
and now we have two threads on the bridge in opposite directions.

————————end solution————————

b. **(6 mark(s))** Show how the declared condition variable and lock could be used to correctly synchronize the cars by annotating and/or modifying the above code with calls to Condition Variable and Lock operations.

————————begin solution————————

```
This part is very similar to the produce / consumer code studied
in class.
```

```
     Arrive(int mydir)
     {
A.1    lock_acquire(lock);
A.2      while (curdir != mydir &&
A.3             curdir != DIR_OPEN)  {
A.4         wait(cv, lock);
A.5         /* REMOVE ; spin */
A.6
A.7      }
A.8
A.9      numcars++;
A.10
A.11     curdir = mydir;
A.12   lock_release(lock);
     }
```

```
     Depart(int mydir)
     {
D.1    lock_acquire(lock);
D.2      numcars--;
D.3
D.4      if (numcars == 0) {
D.5
D.6        dir = DIR_OPEN;
D.7        cv_broadcast(cv, lock);
D.8      }
D.9
D.10  lock_release(lock);
D.11
D.12
      }
```

————————end solution————————

c. **(2 mark(s))** In your solution, draw arrows connecting the start and end of all the critical sections of code.

————————begin solution————————

```
As shown/discussed in class using producer / consumer example \\
Lines A.1 -> A.4, A.1 -> A.12, A.4 -> A.12 and D.1 -> D.10
```

————————end solution————————

continued on the next page ...

d. **(2 mark(s))** Can your solution lead to starvation? Briefly explain.

——————begin solution——————

Yes. Once cars start moving in one direction, they will starve out cars waiting to go in the other direction indefinitely (as long as more cars continue to arrive in the first direction).

——————end solution——————

e. **(2 mark(s))** Can your solution lead to deadlock? Briefly explain.

——————begin solution——————

No, not all of the deadlock conditions are met (e.g., no circular wait). In specific terms, for example, there is no way for cars going in both directions to wait at the same time.

——————end solution——————

The space below is intentionally left blank.

Problem 4 (10 marks)

CHOOSE AND ANSWER ONLY TWO OF THE THREE PARTS OF THIS QUESTION. IF YOU ANSWER ALL THREE PARTS, THE LOWEST TWO SCORES WILL BE USED.

Some useful info: $2^{10} = 1$ KB, $2^{20} = 1$ MB, $2^{30} = 1$ GB

a. **(5 mark(s))** In this part of the question all addresses, virtual page numbers and physical frame numbers are represented in hexidecimal, recall that each hexidecimal character represents 4 bits. Consider a machine with *48-bit* virtual addresses and a page size of 16 MB. During a program execution the TLB contains the following valid entries (in hexidecimal).

Virtual Page Num	Physical Frame Num
0x 0	0x 1
0x BB	0x 2
0x BB9	0x 3
0x BB97	0x 4
0x BB972	0x 5
0x BB9720	0x 6

Translate the following virtual address into a *52-bit* physical address (in hexidecimal). Show and explain how you derived your answer. Express the final physical address using all 52-bits. Virtual address = 0x 00BB 9720 AF05.

——————begin solution——————

16 MB page size = 2^{24} so 6 hex digits for offset.
48-24 = 24 bits (6 hex digits for virtual page number.

```
0x 00BB 97|20 AF05
Lookup 00BB97 in TLB, match and physical frame = 4 so
physical addr => 0x 000 0004 | 20AF05 (NOTE: 52-bit result).

NOTE: the solution used (and unforunately NOT CHECKED) by the
TAs used a 56-bits physical addr. So some people may have
lost one mark here even though the solution was correct.
Others who use 56-bits may have been given an extra mark
even though it was incorrect. We apologize to both groups ;-)
```

——————end solution——————

b. **(5 mark(s))** In this part of the question all addresses, virtual page numbers and physical frame numbers are represented in octal, recall that each octal character represents 3 bits. Consider a machine with *24-bit* virtual addresses and a page size of 512 bytes. During a program execution the TLB contains the following valid entries (all in octal).

Virtual Page Num	Physical Frame Num
0	10
7	20
73	30
731	40
7310	50

Translate the following virtual address (in octal) into a *24-bit* physical address (in octal). Show and explain how you derived your answer. Express the final physical address using all 24-bits. Virtual address = 07310525.

—————begin solution—————

$2^9 = 512$ so 9 bits for offset. 24-9 = 15 for VPN.
9 bits = 3 octal characters, 15 bits = 5 octal characters.
So the first 5 octal characters are the VPN and the last 3 are the offset

```
07310 matches entry in TLB and frame num = 50. So
07310|525 => physical addr 00050|525
```

—————end solution—————

c. **(5 mark(s))** In this part of the question all addresses, virtual page numbers and physical frame numbers are represented in decimal. Consider a machine with *32-bit* virtual addresses and a page size of 512 bytes. During a program execution the TLB contains the following valid entries (all in decimal).

Virtual Page Num	Physical Frame Num
591	100
5912	200
11548	300
2589	400
59125	500

Translate the following virtual address (in decimal) into a physical address (in decimal). Show and explain how you derived your answer. Express the final physical address using decimal digits only. Virtual address = 5912589.

————————begin solution————————

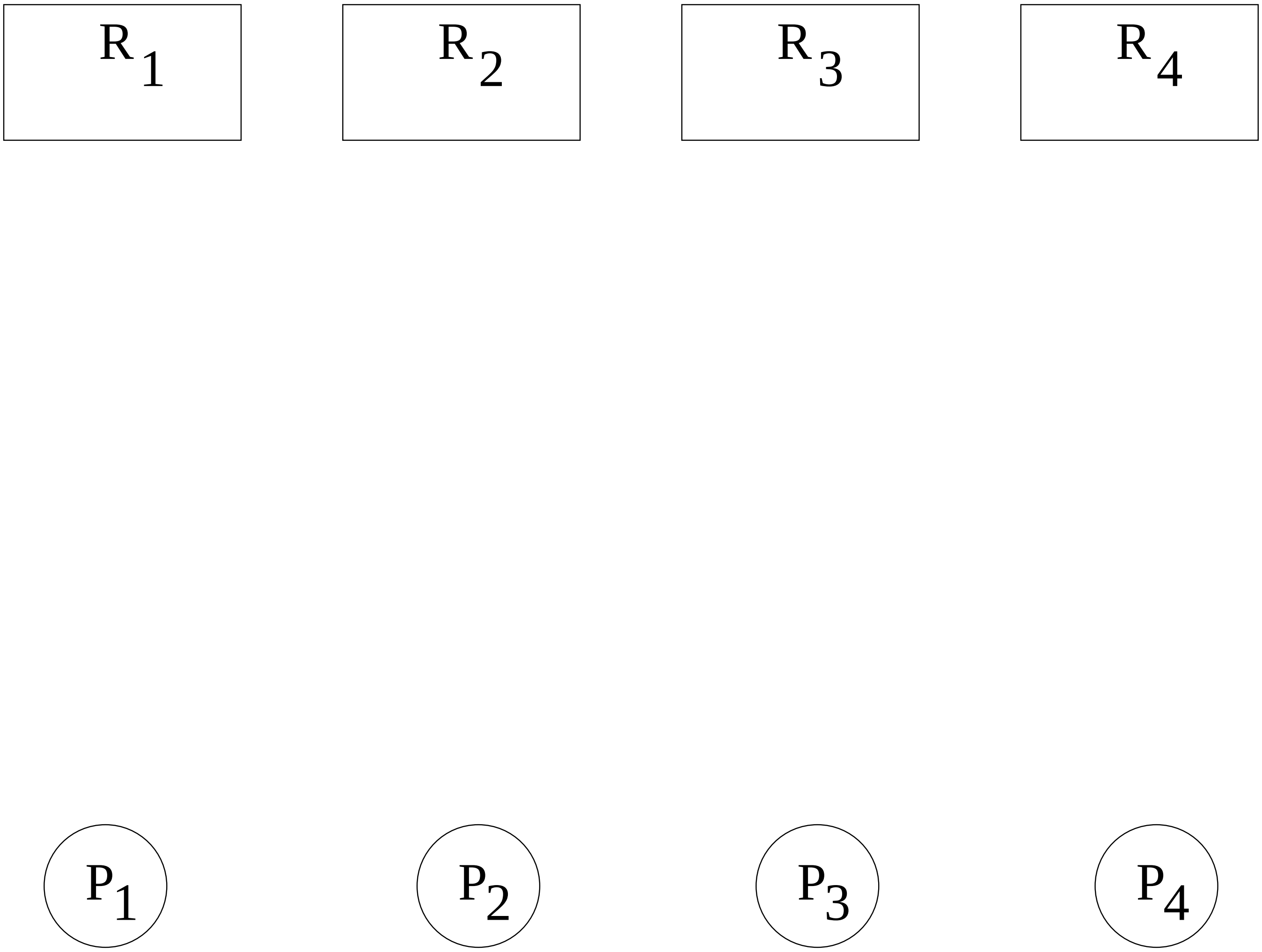
```
Showing the work for the long division.

VPN = floor(vaddr / pagesize)
offset = vaddr % pagesize

        11548   <= VPN

     ---------------
512 | 5912589
      512
       792
       512
       2805
       2560
        2458
        2048
         4109
         4096
           13  <= Offset

Lookup 11548 in TLB, match found = 300 so frame number = 300.
Now compute the address of the start of that frame.
    512 Pagesize
    300 Frame
 ------
 153600 <= Start of the frame the address is found in
  +  13 <= Add the offset
 ------
 153613 <= This is the physical address.
```

————————end solution————————

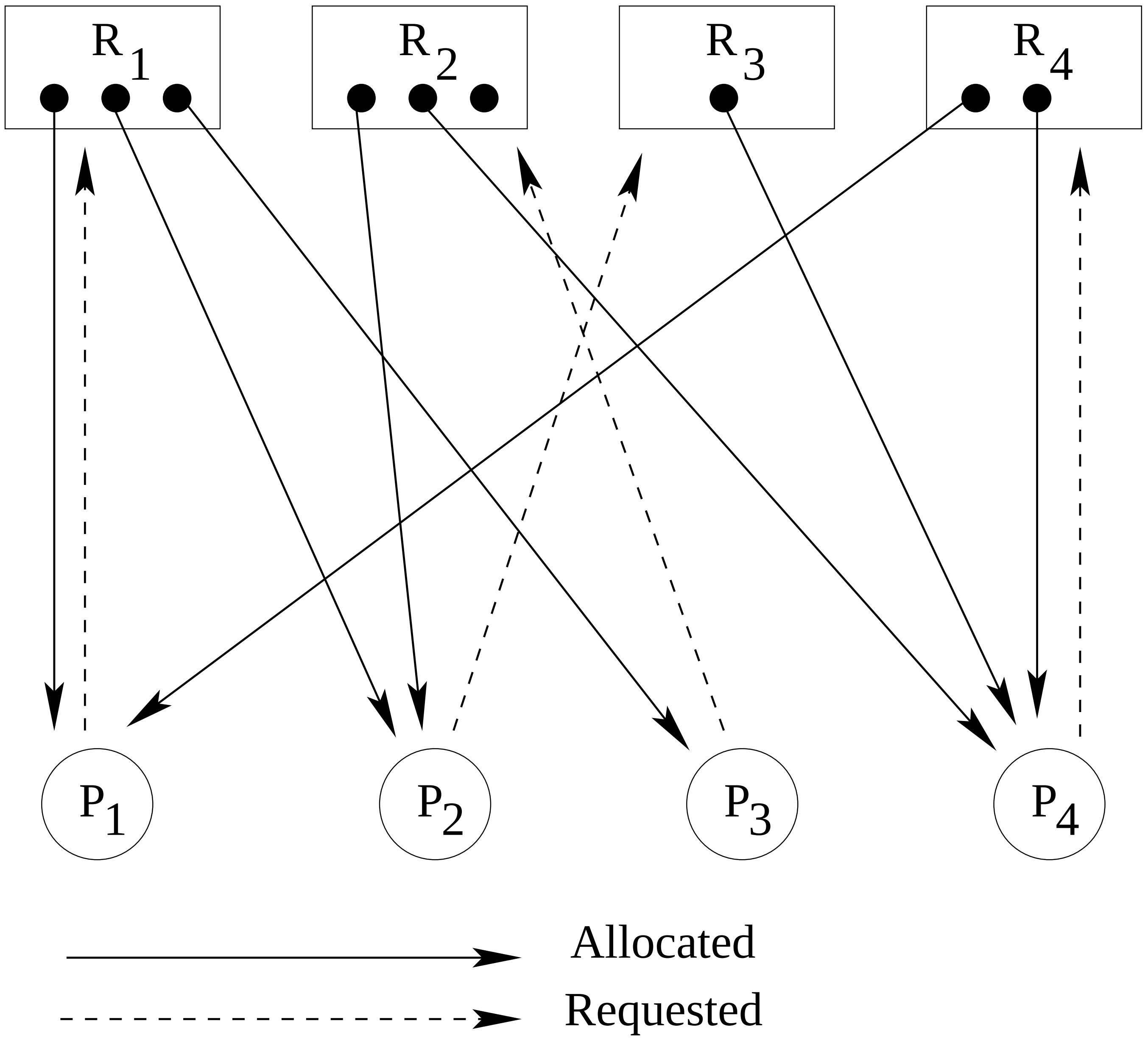
Problem 5 (9 marks)
This question uses the following notation (as used in the course notes) to describe resource allocation in a computer system :

- R_i: request vector for process P_i
- A_i: current allocation vector for process P_i
- U: unallocated (available) resource vector

Given the scenarios below, draw the corresponding resource allocation graph. PLEASE USE SOLID LINES WITH ARROWS FOR ALLOCATION EDGES AND DASHED OR DOTTED LINES WITH ARROWS FOR REQUEST EDGES. Indicate if the system is deadlocked and justify your answer (one sentence).
$U = (0,1,0,0)$
$R_1 = (1,0,0,0)$, $R_2 = (0,0,1,0)$, $R_3 = (0,1,0,0)$, $R_4 = (0,0,0,1)$
$A_1 = (1,0,0,1)$, $A_2 = (1,1,0,0)$, $A_3 = (1,0,0,0)$, $A_4 = (0,1,1,1)$.



begin solution



The system is NOT deadlocked.

P_3 needs and R_2 and one is unallocated so assign it to P_3 then, it (P_3) has all the resourse it needs so it finishes and free $1xR_1$,
P_1 gets R_1 and then it can finish and free $2xR_1 and 1xR_4$
P_4 gets R_4 and then it can finish and free $2xR_4 and 1xR_3$
P_2 gets R_3 and then it can finish and free $1xR_1 and 1xR_2 and 1xR_3$

Common mistakes were not properly drawing the edges and not properly justifying the answer.

——————end solution——————

Problem 6 (6 marks)

a. **(3 mark(s))** Write a small C code fragment that when run on OS/161 would generate an "Address Error on Load" exception (USING AN ADDRESS THAT IS NOT PART OF THE KERNEL'S ADDRESS SPACE). Comment your code to explain what it is doing.

————begin solution————

```
NOTE: There are quite a lot of possibilities here.
The key will be getting a proper address that is
not part of the processes address space.
One possiblity is:

char *a = 0x0;  /* create a pointer to an invalid address : 0 */
char b;
b = *a;         /* now try to access that address (requires a load) */


Some other examples:
char *a =  // some value less than the start of the code segment
              e.g., < 0x0040 0000
char *a =  // some value between the code segment and data segment
              e.g., > 0x1001 0000 and < 0x7FFF 4000
char *a =  // some value between the data segment and the stack segment
              e.g., > 0x1001 0000 and < 0x7FFF 4000

Any address that is part of the address space is incorrect.
Any address that is part of the kernel's address space is incorrect.
  i.e., >= 0x8000 0000
```

————end solution————

b. **(3 mark(s))** Write a small mips assembly code fragment that makes an OS/161 system call with the value 57 as an argument to the system call numbered 7. You are not required to worry about the return value of the system call, to have correct syntax, or recall all instruction names precisely; pseudo-assembly language is fine. Comment your code to explain what it is doing.

————begin solution————

```
li a0 57        // Put 57 into the register that is used as the first argument
li v0 7         // Put the system call number into v0 (used to indicate which syscall)
syscall         // Make the system call now
```

————end solution————