CS 342302 Operating Systems

Fall Semester 2021

Prof. Pai H. Chou

Weekly Review 4

The questions here serve the purpose of reviewing concepts from the lecture, and expect the concepts to be tested on the midterm and final. However, they are by no means exhaustive. Anything covered in the lecture and projects can be tested.

1. Definitions and Short Answers - week 4 (10/4 lectures)

1. What is the difference between a **program** and a **process**? Which one is active and which one is passive?

program: executable code

process: an instance of a program in execution

1. What is a **job**?

=process

1. What is a **task**?

informal word for process

1. What is the **text section** of a process's memory?

binary code segment

1. What is the difference between the **data** section and **heap** section of a process's memory?

data: global vars

heap: dynamically allocated var, including objects

1. What is kept in the **stack** section of a process's memory?

local vars of functions, parameters passed to function call, return address

1. Besides the content of the main memory in a process's address space, what other **state** does a process have (that is maintained by the OS)?

program counter, registers

1. What are the **five states** of a process (in the textbook's terminology)?

new, ready, running, waiting, terminated

1. On a single processor, how many processes can be in **running** state at a time? in **ready** state? in **waiting** state?

1, some, some

1. Why would a process enter **waiting** state?

waiting for service to complete(IO or request)

1. From which three states may a process **transition to ready** state?

new, running, waiting

1. How can a process transition from **running to ready** state without doing any system call?

timer interrupt

1. In the process control block (PCB), what is the meaning of the following fields?
   1. **program counter**

where you are in the program

* 1. **process number**

uniquely indentify the process

* 1. CPU registers

store temporary values

* 1. memory management information

memory allocated to the process

* 1. accounting information

CPU used, clock time elapsed since start, time limits

* 1. I/O status information

IO devices allocated to process, list of open files

1. What is a **thread** of execution?

lightweight processes

1. On a single(-core) processor,
   1. how many **processes** can be **running** at a time?

1

* 1. how many **threads** can be **running** at a time?

some

1. On a multiprocessor system,
   1. how many processes can be running at a time?

some

* 1. how many threads can be running at a time?

some

1. What is the definition of the **degree of multiprogramming**? Does it reflect
   1. the number of processors in the system?
   2. the number of processes currently running, waiting, ready?
   3. the number of threads currently running, waiting, ready?
   4. the amount of main memory in the system?
   5. the scheduling policy?

number of process kept in memory

1. What is the difference between an **I/O-bound** and a **CPU-bound** process?

I/O-bound: spends more time doing IO than computing

CPU-bound: spends more time doing computation

1. What does **context switching** mean?

switch to a different process to run

1. What are the possible triggers for context switch?

interrupt, system call

1. What are the steps taken by the kernel to switch from process P1 to process P2?

save P1 state into PCB1, restore P2 state from PCB2

1. What kinds of support can hardware provide to help reduce the overhead of context switching?

instruction for store/load multiple registers, register windows

1. What is a **pid**?

a unique identifier for each process

1. In a Unix-like system, how is fork() used for creating a new process?

parent clone itself

1. After fork() creates the child process, where in the program does the child process start running?

child and parent execute concurrently

1. What is the meaning of the return value of fork()?

child get 0, parent get pid of child

1. What does exec() do? Does it create a new process? Does exec() return?

replaces process itself with specified program

no, restart process

if success, doesnt return. if error, return -1 with error code in global var “errno”

1. How does a shell launch a program as a new process using fork() and exec()?

calls fork( ): create new process for new program

child process call exec( ): load in new program, becomes new program

1. What is **copy-on-write**? How does it improve the efficiency of the original implementation of fork()?

no need to store extra copy of the same data

saves work of copying

1. How many processes are created by the following example? Explain  
   #include <stdio.h>  
   #include <unistd.h>  
   int main() {  
    for (int i=0; i<3; i++) {  
    fork();  
    }  
    return 0;  
    }

7

1. How does a process terminate voluntarily?

exit(status), implicit exit upon return from main

1. What are three reasons a child process may be terminated?

voluntary, involuntary(killed), OS clean-up

1. What does **cascaded termination** mean?

kill parent, children have to be killed too

1. What is an **orphan** process? a **zombie** process?

a child process alive whose parent died

dead child process that died before its parent calls wait() to find out

1. What happens if the parent of a zombie process does not call wait()?

zombie pid wont be released

1. What happens on termination of the orphan processes whose parent died before it had a chance to call wait() on its children? What is a solution?

the OS wont recycle that process id

an ancestor process could call wait() to collect orphans

1. What is the name of the "root" process in traditional Unix, with a pid of 1? What is the Linux version called?

init

systemd

1. What is the main advantage of shared memory communication over message passing and why? What must be performed carefully for shared memory to work consistently?

faster performance

need to determine the form of data and location

1. What are some ways to ensure shared data are not written simultaneously in inconsistent ways?

synchronize by locking or scheduling

1. When is shared memory not an option for two processes to communicate?

across machine

1. What does it mean that message passing calls such as send() or receive() may **block**?

call something and it doesnt return to you until it is done

1. In the pseudocode for shared memory communication, what are the purposes of variables named in and out? Which variable is modified by consumer and which is modified by producer? Can both the sender and receiver be modifying the same variable at the same time, assuming they can run at the same time?

2 pointers to remember the position

out, in

no

1. What is "direct communication" (style of message passing) between two processes? How many processes can a link be associated with? How many links can exist between a given pair of processes?

process must name each other explicitly

2

1

1. In direct communication, how is receive() in asymmetric naming different from receive() in symmetric naming? Can more than one sender process send messages to a given receiver process? If so, can the receiver know the identity of the sender and how?

asymmetric: dont name sender, symmetric: name sender

yes

yes, by sender id

1. Why does direct communication have the **limited modularity** problem? How is it solved by indirect communication?

since process must name each other, so if name of a process changed, all old names need to be updated

dont name each other

1. Does indirect communication allow one sender and multiple receivers to share the same mailbox?

yes

1. Is a **synchronous** call (to a function or procedure) blocking or nonblocking? What about an **asynchronous** call?

blocking, nonblocking

1. How does a nonblocking sender/receiver know when the communication is completed (i.e., data has been sent or data is ready to be received)?

a separate call to check if done(like polling)

use a callback for notification

1. What is **rendezvous** communication? Are its sender and receiver blocking and nonblocking? How much buffer is required, if any? How do you pronounce "rendezvous"?

whoever gets there wait for the other one

both blocking

0 buffer

1. in **bounded buffer** communication, under what buffer condition does the sender block and does the receiver block?

if buffer is full => sender block

if buffer is empty => receiver block

1. in **infinite buffer** communication, under what buffer condition does the sender block and does the receiver block?

sender never blocks

receiver blocks only if buffer empty

1. what does RPC mean?

remote procedure calls

1. what is a **stub** function for an RPC on the **client** side, and what does it have to do?

packs params into message

* calls OS to send directly to server
* waits for result to return from server

1. What is a **stub** function on the **server** side, and what does it have to do?

receives call from a client, unpack param

* calls the corresponding procedure
* returns results to the caller

1. What do **marshaling** and **demarshaling** mean during an RPC call? Why can't raw data be sent in their original binary representation? What are example types that cause problems?

marshalling: packs param into a message

unmarshalling: unpacks param

different data representation, char encoding

int, float

1. Why do pointers cause problems when passed as parameters or returned in RPC?

remote call can not access memory on different machine

1. Can RPC always succeed? What kind of problems can happen?

no

connection error

2. Python Programming

1. Write a *generator* for powers of 2:  
   def gen\_powers\_of\_2():  
    # your code here  
   such that if you test it interactively,  
   >>> g = gen\_powers\_of\_2()  
   >>> next(g)  
   2  
   >>> next(g)  
   4  
   >>> next(g)  
   8  
   >>> [next(g) for i in range(5)]  
   [16, 32, 64, 128, 256]
2. Write a generator for elements of a binary tree in **post-order** (i.e., left child recursively, right child recursively, root). Assume the same kind of tree representation as on slide 11: T = (17, (12, (6, None, None), (14, None, None)), (35, (32, None, None), (40, None, None))) for the tree
3. Rewrite the make\_item() generator on slide 21 to eliminate the for c loop. Hint: slide 13.
4. The consumer() on slide 22 is a "pull"-style communication because the consumer asks for the next item (implicitly called by the for loop) from the generator, which does a yield of the next value. Rewrite the **producer() as a function** and **consumer() as a generator** such that the producer "pushes" the items by calling the send(v) method (slide 23) to send value v to the consumer, which receives the value from yield (as an expression) and calls the use\_item() function on each value. In this case, yield just serves the purpose of "receive".  
   def producer():  
    # basically the same as make\_item() on slide 21, except  
    # 1. instantiate revised consumer as a generator g, and  
    # 2. call next() on g once to kickstart it  
    # 3. inside loop, yield c is replaced by a g.send(v)  
   def consumer():  
    # 1. inifinite loop,  
    # 1.1 c = yield, to receive the value sent by producer,  
    # 1.2 call use\_item(c), found on slide 27

4. Sockets programming in Python

Make a minimal web server at port 8086 in Python by displaying the browser's HTTP request header received and then reply with the HTTP reply header and a minimal page in HTML. Use the template shown on slide 34 and print the request; however, since the request is in bytes type instead of str type, To convert to str for printing, you can call the .decode('utf8') method on the bytes data. The HTTP response should look like this:

HTTP/1.1 200 OK

Date: *fill in your own date time, like unix date command*

Server: Python *or call it anything you like*

Last-Modified: *fill in your own date time*

Content-Length: length of

Content-Type: text/html

Connection: Closed

<html>

<body>

<h1>Hello world</h1>

</body>

</html>

Start your web server from the command line

$ python3 webserv.py

Open your own browser to localhost:8086, and it should render the HTML as a web page.