CS351 Final Exam - Fall 2020

The name, username and photo associated with your Google account will be recorded when you upload files and submit this form. Not **twong10@hawk.iit.edu**? Switch account

* Required

Exam Questions

For the programming problems, you must edit file matrixadd.c and runtime.c. You must also write a Makefile to help you compile your code; you should have a single Makefile for all of your programs in this exam. You must check in your code and Makefile to the gitlab repository before the end of the exam. You must copy and paste the contents of your implementation files (matrixadd.c and runtime.c) in the text boxes of this form for archival purposes. However, the programming problems will be graded from the gitlab repository, so you must check your code changes into the repo to receive credit for the programming assignments. You will not receive full credit for your programming component if you do not check in your edits through GIT. Below you will find a list of commands you will be able to use to complete your programming component. You should use fourier.cs.iit.edu as a test system before committing your code.

```
git pull
cd finalexam
git add Makefile
git add matrixadd.c
git add runtime.c
git add finalexam.pdf
git commit -m "finished final exam"
git push
```

Good luck!

(2 points) Which system call may result in the deallocation of an open file description object? 2 points

- [ ] a. open
- [ ] b. read
- [x] c. exit
- [ ] d. dup

Clear selection

(2 points) Which system call may result in the deallocation of an open file description object? 2 points

- [x] a. reducing the number of I/O system calls
- [ ] b. eliminating the possibility of short counts
- [ ] c. supporting the standard input/output/error abstraction
- [ ] d. allowing processes to share files without going through the kernel

Clear selection

(2 points) What is a strong argument against using regular files for dynamic IPC? 2 points

- ( ) a. separate processes cannot simultaneously read/write regular files
- ( ) b. regular files are more prone to short counts (than purpose-built IPC mechanisms)
- (•) c. coordinating separate (e.g., read/write) positions in regular files is tricky
- ( ) d. data stored in regular files does not persist after a process exits

Clear selection

(2 points) What things does the inode data structure not track in a filesystem? 2 points

- ( ) a. Ownership
- ( ) b. Permissions
- ( ) c. Size
- (•) d. Fragmentation
- ( ) e. Type
- ( ) f. Location

Clear selection

(2 points) What are some possible reasons why some short counts can occur in I/O? 2 points

- a. EOF
- b. Unreadable FD
- c. Interrupt
- d. Out of space
- e. All of the above

Clear selection

(2 points) Select which IPC mechanism is appropriate across machine boundaries? 2 points

- a. Signals
- b. Files
- c. Shared memory
- d. Pipes
- e. Semaphores
- f. Atomics
- g. Sockets

Clear selection

(2 points) Which synchronization mechanisms is the most scalable under low concurrency? 2 points

- a. Mutex
- b. Semaphore
- c. Spinlock
- d. Atomic

Clear selection

(2 points) Which synchronization mechanisms is the most scalable under high concurrency? 2 points

- a. Mutex
- b. Semaphore
- c. Spinlock
- d. Atomic

Clear selection

(2 points) What type of storage offers the fastest speed? 2 points

- a. SRAM
- b. DRAM
- c. NVRAM
- d. NVMe SSD
- e. SATA HDD

Clear selection

(2 points) What type of storage offers the best price per byte? 2 points

- a. SRAM
- b. DRAM
- c. NVRAM
- d. NVMe SSD
- e. SATA HDD (selected)

Clear selection

(2 points) Cost of 1TB of storage in 2020 using the cheapest possible technology? 2 points

- a. $35 (selected)
- b. $100
- c. $130
- d. $1103
- e. $7968

Clear selection

(2 points) When performing naïve matrix multiplication (as you did in Lab #5), memory access is sub-optimal due to how the data is stored in memory and how it is accessed in the program. What simple transformatino can be done on the matrix to significantly speedup the performance of the matrix multiplication?

2 points

- [x] a. Transpose
- [ ] b. Compress
- [ ] c. Prime read before compute
- [ ] d. Nothing can be done

Clear selection

(2 points) Which locality of reference does not exist?

2 points

- [ ] a. Temporal
- [ ] b. Spatial
- [x] c. Atomic
- [ ] d. Nothing can be done

Clear selection

(2 points) Which of the following interprocess communication mechanisms is worst suited for the synchronization of multiple processes from a performance point of view?

2 points

- ○ a. shared memory
- ○ b. semaphores
- ○ c. named pipes
- ◉ d. file locks

Clear selection

(2 points) In a cache which resides at the uppermost level of the memory hierarchy (i.e., just below the CPU/registers), we prioritize:

2 points

- ◉ a. improving the hit rate
- ○ b. minimizing the hit time
- ○ c. the implementation of complex replacement policies
- ○ d. high amounts of associativity

Clear selection

What would be the output of the below program (from parent and child processes combined)? 2 points

- [x] a) CS351
- [ ] b) CS351 CS351
- [ ] c) CS351 class
- [ ] d) No output is produced

Clear selection

read_write function

For this and the next question, consider the following function, read_write, which makes use of the buffered stdio function fread:

```
void read_write(FILE *stream, int n) {
  char buf[100];
  /* read n bytes from stream into buf */
  int nread = fread(buf, 1, n, stream);
  /* print bytes read to stdout */
  write(1, buf, nread);
}
```

Note that fread takes three arguments in addition to the destination buffer: - the size of each item to read (1 byte, in the given invocation)
- the number of items to read (n)
- the stream from which to read the items

Assume that stream buffers are 4KB large.
Now consider the following program, which contains two separate calls to read_write:

```
main() {
  FILE *infile = fopen("foo.txt", "r");
  if (fork() == 0) {
    read_write(infile, 6);
  } else {
     wait(NULL);
    read_write(infile, 6);
  }
}
```

Given that the file "foo.txt" contains the single line of text:

```
CS351 class rocks today
```

What is the output of the below program?

2 points

- a) CS351 class
- b) CS351 class rocks
- c) CS351 class class
- d) CS351 class CS351

Clear selection

read_write function followup

Based on the same read_write function and "foo.txt" file from the previous problem, we make a minor modification to the program as shown below, "priming" the stream with yet another call to read_write:

```
main() {
   FILE *infile = fopen("foo.txt", "r");
   read_write(infile,6); /* initial read/write */
   if (fork() == 0) {
     read_write(infile, 6);
   } else {
      wait(NULL);
     read_write(infile, 6);
   }
}
```

(4 points) What is the approximate best case hit rate for loop #1 in the program below? 4 points

- ○ 0%
- ○ 25%
- ○ 50%
- ◉ 75%
- ○ 100%

Clear selection

For this and the next three questions, consider the following function which takes pointers to two non-overlapping 2D arrays (arr1, arr2) of random, word-sized elements, and the number (n) of elements in each to be processed:

```
int foo (int** arr1, int** arr2, int n)
    {

int i, j, accum;

        for (i=0; i<n; i++)   /* loop #1 */
            for (j=0; j<n; j++)
                accum += arr1[i][j];
        for (i=0; i<n; i++)   /* loop #2 */
            for (j=0; j<n; j++)
                accum += arr2[j][i];
        return accum;
    }
```

Make the following assumptions:
- cache size is N-words
- cache lines are 4-words
- all local variables (excluding arrays) are mapped to registers by the compiler
- data in arr1 and arr2 are uncached before foo is called

(4 points) What is the approximate best case hit rate for loop #2 in the above program?

4 points

- [ ] 0%
- [ ] 25%
- [ ] 50%
- [ ] 75%
- [x] 100%

Clear selection

(2 points) What is a primary justification for enforcing alignment of data in memory?

2 points

- [ ] improving cache utilization
- [ ] reducing memory fragmentation
- [x] improving memory (DRAM) utilization
- [ ] simplifying compilation

Clear selection

(4 points) What is the maximum aggregate payload (Pmax) in the program below?

4 points

- a. 160
- b. 300
- c. 335
- d. 475

Clear selection

Given the following sequence of calls to malloc and free:

```
void *p1, *p2, *p3, *p4, *p5, *p6, *p7;
p1 = malloc(40);
p2 = malloc(85);
p3 = malloc(30);
free(p2);
p4 = malloc(20);
free(p3);
p5 = malloc(200);
free(p1);
free(p4);
p6 = malloc(20);
p7 = malloc(80);
```

(4 points) Given a final heap size of 1024 bytes, what is the peak memory utilization in the program above?

4 points

- a. 1024 – Pmax
- b. Pmax/(1024 – 475)
- c. Pmax/(Pmax + 1024)
- d. Pmax/1024

Clear selection

(22 points) True | False

22 points

	True	False
All I/O mechanisms have overlapping requirements that include some form of read/write operations	◉	○
Block I/O is accessed in variable-size chunks supporting seeking and random access	○	◉
Char-by-char streaming access allows seeking and random access	○	◉
Disks are considered to be a block I/O device	◉	○
Networks are considered to be a character I/O device	◉	○
The filesystem acts as a namespace for data residing on different devices	◉	○
"Hello World" is an example of binary data	○	◉
For each process, the kernel maintains a table of pointers to its open file structures	◉	○
File descriptor 0 is	○	◉

reserved to write to standard output		
After opening a file, all file operations are performed using file descriptors	◉	○
A process does not inherit its parent's open files across a fork	○	◉
A process typically asks an OS to write k bytes, but only l<k bytes are actually written	◉	○
The kernel objective in implementing read/write I/O is to support the minimum performance and maximum latency	○	◉
Stream buffers can absorb multiple writes before being flushed to the underlying file	◉	○
When using pipes I/O, the kernel takes care of buffering and synchronization	◉	○
TCP/IP can be used to communicate reliably between processes on the same computer	◉	○
UDP/IP can be used to establish a connectionless communication pattern	◉	○

between processes on different computers		
SRAM has relative speeds in the 1~10 cycles on a modern processor	◉	○
DRAM has relative speeds in the 1000 cycles range on a modern processor	◉	○
HDD has relative speeds in the 10000 cycles range on a modern processor	○	◉
A cache is a store of data for future use	◉	○
Spatial locality is a time-based locality in caching	○	◉

Clear selection

(16 points) Matrix Add

16 points

Implement a matrix add computation of a 2D matrix of size N*N (with random double values between 0.0 and 1.0) added to another 2D matrix of the same size (also with random values between 0.0 and 1.0). The size of the matrix must be set through a command line argument, and must work for any value of N as long as there is enough memory in the system. The function declarations are included, and they must be followed exactly to receive full points. All your code should be contained in the matrixadd.c file, and the Makefile.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <time.h>
#include <math.h>
#include <sys/time.h>
#include <stdbool.h>

#define MSG "running matrixadd with size %s...\n"

#define USAGE "usage: ./matrixadd <size> \n" \
"     - size: 10 / 100 / 1000 / 10000 \n" \


// This function adds mat1[][] and mat2[][],
// and stores the result in res[][]
void add(double** mat1, double** mat2, double** res, int size)
{
	for (int i = 0; i < size; i++) {
		for (int j = 0; j < size; j++) {
			res[i][j] = mat1[i][j] + mat2[i][j];
		}
	}
}

// This function finds the minimum value in res[][] array
```

```
double min(double** res, int size)
{
	double minimum = res[0][0];
	for (int i = 0; i < size; i++) {
		for )int j = 0; j < size; j++) {
			if (res[i][j] < minimum) {
				minimum = res[i][j];
			}
		}
	}
	return minimum;
}

// This function finds the average value in res[][] array
double aver(double** res, int size)
{
	double tot = 0.0;
	for (int i = 0; i < size; i++) {
		for (int j = 0; j < size; j++) {
			tot += res[i][j];
		}
	}
	return tot/size;
}

// This function finds the maximum value in res[][] array
double max(double** res, int size)
{
	double maximum = res[0][0];
	for (int i = 0; i < size; i++) {
		for (int j = 0; j<size; j++) {
			if (res[i][j] > maximum) {
				maximum = res[i][j];
			}
		}
	}
```

```
        return maximum;
}

int main(int argc, char **argv)
{
        time_t t;
        srand((unsigned) time(&t));
    if (argc != 2)
    {
        printf(USAGE);
        exit(1);
    }
    else
    {
        printf(MSG, argv[1]);
        int size = atoi(argv[1]);
                struct timeval start, end;
                size_t len = 0;

                //declare arr1, arr2, and arr3
                double **arr1 = NULL, **arr2 = NULL, **arr3 = NULL;
                //allocate memory for arr1, arr2, and arr3 as a 2D array with
size provided by command line argument
        printf("allocating %lf GB memory...\n",len*3.0/(1024*1024*1024));
                //initialize arr1 and arr2 to random double values between 0
and 1

                //get start timestamp
                gettimeofday(&start, NULL);

                printf("add arr1 and arr2 and store it in arr3\n");
                add(arr1,arr2,arr3,size);
                printf("min(arr3)=%lf\n",min(arr3,size));
                printf("aver(arr3)=%lf\n",aver(arr3,size));
                printf("max(arr3)=%lf\n",max(arr3,size));
```

```
        //get end timestamp
         gettimeofday(&end, NULL);

        double elapsed_time_us = ((end.tv_sec * 1000000 +
end.tv_usec) - (start.tv_sec * 1000000 + start.tv_usec));
        printf("matrixadd with size %d ==> %lf
sec\n",size,elapsed_time_us/1000000.0);
  }
```

Matrix add usage and output

Usage:

```
./matrixadd <size>
```

For example, if you type:

```
$ make
gcc -Wall -O3 -o matrixadd matrixadd.c
$ make test
./matrixadd 10000
running matrixadd with size 10000...
allocating 0.000000 GB memory...
add arr1 and arr2 and store it in arr3
min(arr3)=0.000000
aver(arr3)=0.000000
max(arr3)=0.000000
matrixadd with size 10000 ==> 0.000006 sec
```

This is what you should see on a clean checkout of the code. After you implement all the components, this is the output you should get:

```
$ make test
./matrixadd 10000
running matrixadd with size 10000...
allocating 2.235398 GB memory...
add arr1 and arr2 and store it in arr3
min(arr3)=0.000187
aver(arr3)=0.999967
max(arr3)=1.999926
matrixadd with size 10000 ==> 0.502605 sec
```

(20 points) Runtime

20 points

Implement the runtime utility that has 3 command line arguments. For example, the above command will execute 1000 tasks of sleep 1 second. The thread pool should be started at the beginning and be reused for all the tasks; do not create and destroy threads per task. The parent thread should wait for all tasks to complete before terminating the thread pool and exiting. If the format of the command is not recognized, an error can be displayed. All your code should be contained in the runtime.c file and the Makefile.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <time.h>
#include <math.h>
#include <sys/time.h>
#include <stdbool.h>

#define MSG "running runtime with %s %s %s...\n"

#define USAGE "usage: ./runtime <THREAD_POOL> <NUM_TASKS>
<SLEEP> \n" \
"    - THREAD_POOL: 1 / 10 \n" \
"    - NUM_TASKS: 1 / 10 / 1000 \n" \
"    - SLEEP: 1 / 10 \n"


int main(int argc, char **argv)
{
      time_t t;
      srand((unsigned) time(&t));
   if (argc != 4)
   {
      printf(USAGE);
      exit(1);
   }
```

```
    else
    {
        printf(MSG, argv[1], argv[2], argv[3]);
        int THREAD_POOL = atoi(argv[1]);
        int NUM_TASKS = atoi(argv[2]);
        int SLEEP = atoi(argv[3]);
                struct timeval start, end;

                //Initialize thread pool of size THREAD_POOL
                printf("Initialize thread pool of size %d\n",THREAD_POOL);

        for (int i = 0; i < THREAD_POOL; i++) {
                //sorry idk this
        }
                //get start timestamp
                gettimeofday(&start, NULL);

                //Running NUM_TASKS sleep tasks where each task sleeps
SLEEP seconds
                printf("Running %d sleep tasks where each task sleeps %d
seconds\n",NUM_TASKS,SLEEP);

                //get end timestamp
                 gettimeofday(&end, NULL);

                double elapsed_time_us = ((end.tv_sec * 1000000 +
end.tv_usec) - (start.tv_sec * 1000000 + start.tv_usec));

                printf("Completed running %d sleep %d second tasks using a
thread pool of size %d in %lf
seconds\n",NUM_TASKS,SLEEP,THREAD_POOL,elapsed_time_us/10000
00.0);
    }
```

Runtime Usage and Output

Implement the runtime utility that has 3 command line arguments:
./runtime <thread_pool> <num_tasks> <sleep>

For example, if you type:
$ make test-runtime
./runtime 10 1000 1
running runtime with 10 1000 1...
Initialize thread pool of size 10
Running 1000 sleep tasks where each task sleeps 1 seconds
Completed running 1000 sleep 1 second tasks using a thread pool of size 10 in 0.000002 seconds

And after you finish your implementation:
./runtime 10 1000 1
running runtime with 10 1000 1...
Initialize thread pool of size 10
Running 1000 sleep tasks where each task sleeps 1 seconds
Completed running 1000 sleep 1 second tasks using a thread pool of size 10 in 100.152 seconds

Are you done with all the questions on the exam? If yes, select yes below, export your exam answers as a PDF file, submit your exam through this Google form, and finally commit PDF file and source code to GIT. *

- [ ] I have exported my answers as a PDF file prior to submitting this form

A copy of your responses will be emailed to twong10@hawk.iit.edu.

Back

Submit