Name: UID#: DirectoryID:

CMSC216: Practice Exam 3B

Spring 2026, University of Maryland

Exam period: 20 minutes
Points available: 40

Problem 1 (15 pts): Nearby is a description of the function `half_file()` and a `main()` which demonstrates its behavior. Complete this function in the space provided.

```
#include "half_file.h"

char *half_file(char *fname,
                size_t max_read_bytes,
                size_t *total_bytes_ptr);
// Reads the first half of bytes in ‘fname‘ and
// returns a pointer to heap-allocated memory
// for it. Rounds down if the number of bytes is
// uneven.  Reads ‘max_read_bytes‘ each time
// input is done unless that would read past the
// halfway mark of the file.  Sets argument
// ‘total_bytes_ptr‘ to the number of bytes read
// which should be half the file size. Uses
// low-level UNIX I/O calls to determine file
// size and perform I/O operations. Prints and
// error and returns NULL if opening fails.

int main(int argc, char *argv[]){
  size_t size; char *fname; char *contents;

  fname = "input1.txt";
  printf("input1.txt is:\n");
  system("cat input1.txt");
  contents = half_file(fname, 8, &size);
  printf("half %s is %lu bytes:\n",fname,size);
  printf("%s\n",contents);
  free(contents);
  // input1.txt is:
  // 110 120 130 140 150 160 170 180
  // half input1.txt is 16 bytes:
  // 110 120 130 140

  fname = "input2.txt";
  printf("input2.txt is:\n");
  system("cat input2.txt");
  contents = half_file(fname, 5, &size);
  printf("half %s is %lu bytes:\n",fname,size);
  printf("%s\n",contents);
  free(contents);
  // input2.txt is:
  // 110 120 130 140 150 160 170 180 190 200
  // half input2.txt is 20 bytes:
  // 110 120 130 140 150

  return 0;
}
```

Problem 2 (10 pts): Nearby is a sample `Makefile / Bakefile`. In the space below, draw a picture of the DAG that would be created from this file showing the dependencies between the targets in it.

```
demo : hello bye
	@ echo Running programs
	./hello
	./bye
	@ echo Done running programs

all : hello bye

hello : hello.o
	gcc -o hello hello.o
	echo Ready to say hello

hello.o : hello.c
	gcc -c hello.c

bye : bye.o
	gcc -o bye bye.o

bye.o : bye.c
	gcc -c bye.c

clean :
	rm -f hello.o bye.o hello bye
```

Problem 3 (10 pts): If the command `bake all` were run in the folder with the above `Bakefile`, what commands would be run to complete the build? Show any valid ordering for the commands.

Problem 4 (5 pts): Describe the difference between "Normal" and "Abnormal" process exits. How does a parent process determine if a child process exits normally after successfully completing, exits normally on encountering an error, or exits abnormally? Give code snippets demonstrating how these cases are detected.