Name: UID#: DirectoryID:

CMSC216: Practice Exam 3A

Spring 2026, University of Maryland

Exam period: 20 minutes
Points available: 40

Problem 1 (10 pts): Examine the code to the right and describe what you expect its output to be. Explain why or why not you would expect to see any specific ordering in the output of the program.

```
#include "headers.h"
int main(){
  for(int i=0; i<5; i++){
    pid_t p = fork();
    if(p  != 0){
      wait(NULL);
      printf("iter %d, %d from %d\n",
             i,getpid(),getppid());
      fflush(stdout); // output now
      break;          // quit loop
    }
  }
  exit(0);
}
```

Problem 2 (10 pts): Separa Tememory is trying to understand how child and parent processes interact and is baffled by the behavior of the nearby `childchange.c` program. It seems to her that on changing the `life` variable, it should print a different value later on. Explain to Separa:

(A) Why the program prints what it does
(B) How parent and child processes can affect one another.

```
// childchange.c
#include "headers.h"
int main(){
  int life = 42;
  pid_t pid = fork();
  if(pid == 0){
    life--;
    return 0;
  }
  else{
    wait(NULL);
  }
  printf("life: %d\n",
         life);
  return 0;
}
// >> gcc childchange.c
// >> ./a.out
// life: 42
//   Why not 41??
```

Problem 3 (15 pts): Nearby is a matrix/vector function which performs poorly. Create a new version of this function that **optimizes the memory access pattern**. Show your code and give a brief description of why the changes you made should improve performance.

```
1  int subcol_BASE(matrix_t mat, vector_t vec) {
2    for(int j=0; j<mat.cols; j++){
3      for(int i=0; i<mat.rows; i++){
4        int elij = MGET(mat,i,j);
5        int veci = VGET(vec,i);
6        elij -= veci;
7        MSET(mat,i,j,elij);
8      }
9    }
10   return 0;
11 }
```

```
int subcol_OPT(matrix_t mat, vector_t vec) {
// YOUR CODE HERE
```

WHY CHANGES IMPROVE PERFORMANCE:

Problem 4 (5 pts): Consider the code sample nearby which prints logging messages to either the screen or a log file as dictated by the `USE_LOGFILE` variable. Describe how one could eliminate the conditional `if/else` and all the `fprintf()` calls using **I/O redirection system calls** within the program.

```
1  {
2    if(USE_LOGFILE){
3      fprintf(logfile,"Updating DB\n");
4    }
5    else{
6      printf("Updating DB\n");
7    }
8    update_db();
9    if(USE_LOGFILE){
10     fprintf(logfile,"Syncing files\n");
11   }
12   else{
13     printf("Syncing files\n");
14   }
15   file_sync();
16   ...;
17 }
```