coe628 Final Study Guide (2017)

Table of Contents

What is a Study Guide?

The study guide gives you some idea of what you should know for the exam.

You should attempt all questions and only look at the answers afterwards.

The study guide is not exhaustive. There will be questions on the final on topics not in this study guide. None of the questions here will be on the final. Topics in the guide will not necessarily be in the final.

The Questions

Address of global variables in forked processes

Given the following program:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
int global = 5;

int main(int argc, char** argv) {
    int local = 5;
    int i;
    int junk;
    if (fork() == 0) {
        printf("CHILD global address: %p\n", &global);
        for (i = 0; i < 10; i++) {
            local++;
            global++;
            printf("CHILD local: %d, global: %d\n", local, global);
        }
        exit(EXIT_SUCCESS);
    }
    wait(&junk);
    printf("PARENT global address: %p\n", &global);

    for (i = 0; i < 10; i++) {
        local--;
        global--;
        printf("PARENT local: %d, global: %d\n", local, global);
    }

    return (EXIT_SUCCESS);
}
```

The output is:

```
CHILD global address: 0x100402010
CHILD local: 6, global: 6
CHILD local: 7, global: 7
CHILD local: 8, global: 8
CHILD local: 9, global: 9
CHILD local: 10, global: 10
CHILD local: 11, global: 11
CHILD local: 12, global: 12
CHILD local: 13, global: 13
CHILD local: 14, global: 14
CHILD local: 15, global: 15
PARENT global address: 0x100402010
PARENT local: 4, global: 4
PARENT local: 3, global: 3
```

```
PARENT local: 2, global: 2
PARENT local: 1, global: 1
PARENT local: 0, global: 0
PARENT local: -1, global: -1
PARENT local: -2, global: -2
PARENT local: -3, global: -3
PARENT local: -4, global: -4
PARENT local: -5, global: -5
```

The **bold** lines in the output indicate that both parent and child have the same address for the variable "global". However, the value of this variable is different for parent and child. Explain this apparent paradox.

Memory spaces

Given the following program:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int g1;
int g2;
char * gcp1;
char * gcp2;

void foo(int p, int q) {
    printf("Parameter p address:   %p\n", &p);
    printf("Parameter q address:   %p\n", &q);
}

int main(int argc, char** argv) {
    int local_1;
    int local_2;

    gcp1 = "abc";
    printf("Function main address:          %08X\n", main);
    printf("Function foo address:           %08X\n", foo);

    printf("Global g1 address:            %p\n", &g1);
    printf("Global g2 address:            %p\n", &g2);
    printf("Global gcp1 address:          %p\n",  &gcp1);
    printf("Global gcp2 address:          %p\n",  &gcp2);

    printf("Local local_1 address:       %p\n", &local_1);
    printf("Local local_2 address:       %p\n", &local_2);
    foo(local_1, 5);

    printf("String 'abc'  address:  %p\n", gcp1);
    gcp2 = (char *) malloc(strlen(gcp1) + 1);
    strcpy(gcp2, gcp1);
    printf("Dynamically allocated copied string address: %p\n", gcp2);
```

```
    *gcp2 = 'x';
    printf("%s\n", gcp2);

//    *gcp1 = 'x';
    printf("Goodbye\n");

    return (EXIT_SUCCESS);
}
```

The output is:

```
Function main address:          00401118
Function foo address:           004010E0
Global g1 address:             0x1004071c0
Global g2 address:             0x1004071d0
Global gcp1 address:           0x1004071c8
Global gcp2 address:           0x1004071d8
Local local_1 address:        0x5fcb0c
Local local_2 address:        0x5fcb08
Parameter p address:   0x5fcae0
Parameter q address:   0x5fcae8
String 'abc'  address:  0x100403066
Dynamically allocated copied string address: 0x6000483f0
xbc
Goodbye
```

a) Identify the names of the memory spaces associated with each of the printed addresses.

b) If the line `*gcp1 = 'x';` is NOT commented out, the output is the same EXCEPT the "Goodbye" line is not printed and the program crashes with the message "`Segmentation fault (core dumped)`"". Explain.

Amdahl's law

An algorithm is 20% serial; the rest can be run in parallel. What is the maximum speedup if 5 processors can be used?

Semaphores

Implement the up and down functions for semaphores in "pseudo-C". Assume that the values of semaphores are maintained in a global array of ints called "sems[]". Assume that the up and down

functions take a single int parameter which is the semaphore id. Also assume that there following functions available:

- sem_notify(int sem_id) which will unblock a process that is waiting for the semaphore.
- sem_wait(int sem_id) which will place the current process in the BLOCKED state and call the kernel scheduler.
- disable() and enable() that enable or disable interrupts.
- Note: assume up() and down() are kernel functions so it is OK to disable interrupts in a critical section.

Banker's algorithm

Consider the following resource-allocation policy. Requests and releases for resources are allowed at any time. If a request for resources cannot be satisfied because the resources are not available, then we check any processes that are blocked, waiting for resources. If they have the desired resources, then these resources are taken away from them and are given to the requesting process. The vector of resources for which the waiting process is waiting is increased to include the resources that were taken away.

For example, consider a system with three resource types and the vector Available initialized to (4,2,2). If process P0 asks for (2,2,1), it gets them. If P1 asks for (1,0,1), it gets them. Then, if P0 asks for (0,0,1), it is blocked (resource not available). If P2 now asks for (2,0,0), it gets the available one (1,0,0) and one that was allocated to P0 (since P0 is blocked). P0's Allocation vector goes down to (1,2,1), and its Need vector goes up to (1,0,1).

a) Can deadlock occur? If so, give an example. If not, which necessary condition cannot occur?

b) Can indefinite blocking occur?

Test and set

Explain how the test-and-set instruction is used.

Dining Philosophers

Explain the dining philosophers problem. Provide an elementary naive algorithm that will possibly result in deadlock. Give a proper algorithm that solves the problem. (Each algorithm may be in pseudo-code or a language of your choice.

Dup/Dup2/Pipe

Peterson's Algorithm

Paging