Chalmers University of Technology and Gothenburg University

Operating Systems
EDA093, DIT 401

Exam 2023-01-03

Date, Time: Tuesday 2023/01/03, 08.30-12.30

Course Responsible:
Vincenzo Gulisano (031 772 61 47)

Auxiliary material: You may have with you
- An English-Swedish, Swedish-English dictionary.
- No other books, notes, calculators, etc.

Grade-scale ("Betygsgränser"):
CTH: 3:a 30-39 p, 4:a 40-49 p, 5:a 50-60 p
GU: Godkänd 30-49p, Väl godkänd 50-60 p

Exam review ("Granskningstid"):
Will be announced after the exam.

Instructions

- Do not forget to write your personal number, if you are a GU or CTH student and at which program ("linje").
- Start answering each assignment on a new page; number the pages and use only one side of each sheet of paper.
- Write in a **clear manner** and **motivate** (explain, justify) your answers. If it is not clear what is written, your answer will be considered wrong. If it is not explained/justified, even a correct answer will get **significantly** lower (possibly zero) marking.
- If you make **any assumptions** in answering any item, do not forget to clearly state what you assume.
- The exam is organized in groups of questions. The credit for each group of questions is mentioned in the beginning of the respective group. Unless otherwise stated, all questions in a group have equal weight.
- Answer questions in English, if possible. If you have large difficulty with that and you think that your grade can be affected, feel free to write in Swedish.

Good luck !!!!

1. (12 p) Present Peterson's algorithm and prove it satisfies the mutual exclusion property.

   [**HINT:** You can find the description of the algorithm as well as the related proofs in the synchronization lecture. Up to 6 points are given based on how you present Peterson's algorithm. Up to 6 points are given based on how you present the requested proof. Please notice the question is not about general discussions on safety properties or other related algorithms, so no points are given based on those discussions.]

2. (12 p)

   (a) (8 p) The following code is intended to print "Greetings" n times, with n being a non-negative number given by a user. Spot all the errors, explain why they are errors, and propose a way to fix each error you spot. Please notice: the code is not intended to perform its task efficiently nor with the smallest number of instructions. The errors the question refers to are actual bugs that prevent the program from doing what it is supposed to do.

```
#define BUFFER_SIZE 25
#define READ_END 0
#define WRITE_END 1

int n; // Number of times "Greetings" should be printed

int main(void)
{
    char write_msg[BUFFER_SIZE] = "Greetings";
    char read_msg[BUFFER_SIZE];
    pid_t pid;
    int fd[2];

    pthread_t tid;
    pthread_attr_t attr;

    if (argc != 2) {
        fprintf(stderr,"usage: a.out <integer value>\n");
        return -1;
    }

    if (atoi(argv[1]<0) {
        fprintf(stderr,"%d must be >=0\n",atoi(argv[1]));
        return -1;
    }

    pthread_attr_init(%attr);
    pthread_create(&tid,&attr,runner,argv[1]);
    pthread_join(tid,NULL);

    /* now fork a child process */
    pid = fork();

    /* create the pipe */
    if (pipe(fd) > 0) {
        fprintf(stderr,"Pipe failed");
        return 1;
    }

```

```
40      if (pid < 0) {
41          fprintf(stderr, "Fork failed");
42          return 1;
43      }
44
45      if (pid > 0) { /* parent process */
46          /* close the unused end of the pipe */
47          close(fd[READ_END]);
48          /* write to the pipe */
49          write(fd[WRITE_END], write_msg, strlen(write_msg)+1);
50          /* close the write end of the pipe */
51          close(fd[WRITE_END]);
52      }
53      else { /* child process */
54          /* close the unused end of the pipe */
55          close(fd[READ_END]);
56          /* read from the pipe */
57          read(fd[WRITE_END], read_msg, BUFFER_SIZE);
58          for (i = 1; i < n; ++i)
59          {
60              printf("%s\n",read_msg);
61          }
62          /* close the write end of the pipe */
63          close(fd[READ_END]);
64      }
65      return 0;
66  }
67
68  void *runner(coid *param)
69  {
70      int n = atoi(param);
71      pthread_exit(0);
72  }
```

[**HINT:** Pipe after fork. Wrong check on pipe result. Child closes/reads from wrong end of the pipe. Runner stored value passed by user in local n instead of the global one.]

(b) (4 p) Can an OS that can only run single-threaded processes still allow for a program to use multiple cores in parallel? Motivate your answer.

[**HINT:** The program can run several processes and these processes can communicate / share-memory.]

3. (12 p)

(a) (4 p) Using the Least Recently Used (LRU) page replacement algorithm and assuming 5 frames, write a reference string that results in 8 page faults, excluding the page faults incurred to initially fill all 5 frames. Show how you compute the page faults / motivate your answer.

[**HINT:** Various strings can be defined. The simplest I can think of would be 1 2 3 4 5 6 7 8 9 10 11 12 13.]

(b) (4 p) Describe how the Copy-on-Write mechanism works.

[**HINT:** Check related slides.]

(c) (4 p) Describe how a lower and an upper bound (with the former being lower than the latter) used to monitor the page-fault frequency can help an OS in detecting thrashing. Explain why and how wrong estimations of such bounds can lead to false detection of thrashing.

[**HINT:** Check related slides.]

4. (12 p)

   (a) (3 p) Assume a multicore computer is being used by dedicating each CPU/core to a different OS instance, and that the main memory of such a computer is partitioned evenly across all OS instances. For simplicity, assume the computer does not have caches. That is, the CPU/cores read/write data to the main memory directly. Provide an example of how two OS instances might end up reading different versions of data coming from the same I/O source because of a lack of synchronization.

   [**HINT:** OS 1 reads file F from the disk. OS 2 reads, modifies, and stores F back to the disk. OS 3 reads file F from the disk.]

   (b) (3 p) Can Shortest Job First (non-preemptive) scheduling result in starvation? Can Shortest Remaining Time Next (preemptive) scheduling result in starvation? Motivate your answers.

   [**HINT:** They both can, please check related slides.]

   (c) (6 p) Prove that the Shortest Job First (non-preemptive) scheduling for a certain set of processes/jobs to be scheduled is optimal in terms of turnaround time (hint: the proof can be made by contradiction, trying to find a different order that results in lower turnaround time).

   [**HINT:** Turnaround time = waiting time + execution time averaged for all jobs. Let's say jobs are sorted on duration. Also, T_i is the turnaround of job i, W_i the waiting time of job i, and D_i the duration of job i. Then:

   i. $T_1 = D_1$
   ii. $T_2 = W_2 + D_2 = D_1 + D_2$
   iii. $T_3 = W_3 + D_3 = D_1 + D_2 + D_3$
   iv. ...

   If any pair is swapped, the sum of turnaround time grows, and so does the average:

   i. $T_1 = D_1$
   ii. $T_3 = W_1 + D_3 = D_1 + D_3$
   iii. $T_2 = W_2 + D_2 = D_1 + D_2 + D_3$
   iv. ...

   ]

5. (12 p)

   (a) (3 p) Explain the difference between Access Control Lists and Capabilities.

   [**HINT:** Check related slides in the Security lecture.]

   (b) (3 p) If an n-bit salt is used for authentication, how many dictionaries would an attacker need to use in the worst case? Motivate your answer.

   [**HINT:** Check related slides in the Security lecture.]

   (c) (6 p) Explain in detail how a buffer overflow attack works.

   [**HINT:** Check related slides in the Security lecture.]