Operating Systems ID1206

(ID2200/06 6hp)

Exam

2019-04-16 14:00-18:00

Instruction

- You are, besides writing material, only allowed to bring one self hand written A4 of notes. The notes are handed in and can not be reused.
- All answers should be written in these pages, use the space allocated after each question to write down your answer.
- Answers should be written in Swedish or English.
- You should hand in the whole exam and the hand written page of notes. No additional pages should be handed in.

Grades

The exam is divided into a number of questions where some are a bit harder than others. The harder questions are marked with a star *points**, and will give you points for the higher grades. The exam is thus divided into basic points and points for higher grades. First of all make sure that you pass the basic points before engaging with the higher points.

Questions with multiple sub-questions are normally awarded 2p for all correct and 1p for one wrong answer.

Note that, of the 12 basic points only at most 11 are counted, the points for higher grades will not make up for lack of basic points. The limits for the grades are as follows:

- Fx: 6 basic points
- E: 7 basic points
- D: 8 basic points
- C: 10 basic points
- B: 11 basic points and 5 higher points
- A: 11 basic points and 8 higher points

The limits could be adjusted to lower values but not raised.

Name:____________________________ Persnr:____________________________

Answer: The provided answers are not necessarily answers that would result in full points, longer explanations could be required.

1 Processer

1.1 stack or heap [2 points]

What is done in the procedure below and where should `gurka` be allocated? Why? Complete the code so that `gurka` is allocated space.

```
int *tomat(int *a, int *b) {
  // allocate room for gurka


  *gurka = *a + *b;
  return gurka;
}
```

Answer: The function will return a pointer to a `int` that then must be allocated on the heap. The variable `gurka` should point to a heap allocate are large enough to store an `int`. This is achieved by:

```
int *gurka = (int*)malloc(sizeof(int));
```

Name:____________________________ Persnr:____________________________

1.2 fork() [2 points]

What is printed when we run the program below, what alternatives exist and why do we get this result?

```
int global = 17;

int main() {
  int pid = fork();
  if(pid == 0) {
    global++;
  } else {
    global++
    wait(NULL);
    printf("global = %d \n", global);
  }
  return 0;
}
```

Answer: The output will be `global = 18` once. The two processes will have their own copy of the data segment and hence `global`. Since the updates are independent of each other the final result in both cases is 18. Only the mother process will output the result.

Name:____________________________ Persnr:____________________________

1.3 `__sync_val_compare_and_swap()` [2 points*]

We can implement a spin lock in GCC as shown below. The lock is implemented using a machine instruction that atomically will read the content of a memory location and, if it is equal to our requirement, replace it with a new value. In the implementation below we represent an open lock with the value 0; if the lock is open we write a 1 in the location and return 0, otherwise we return the value found (that then should be 1).

Assume that we use the lock to synchronize two threads on a machine with only one core; what is then the disadvantage that we will have? How could we mitigate the problem?

```
int try(volatile int *mutex) {
  return __sync_val_compare_and_swap(mutex, 0, 1);
}

void lock(volatile int *mutex) {
  while(try(mutex) != 0) { }
}

void release(volatile int *mutex) {
  *mutex = 0;
}
```

Answer: If a thread takes the lock and is then suspended by the scheduler, the scheduled thread could in the worst case spend its whole allotted time slot spinning. We could yield the CPU, `pthread\_yield()`, to allow the suspended thread to continue its execution.

Name:____________________________ Persnr:____________________________

2 Communication

2.1 from one to the ...[2 points]

Assume that we have two programs, `ones` and `add2`, implemented as bellow. The call to `scanf("\%d", \&in)` will read from `stdin` and parse a number that is then stored in `\&in`. The procedure either returns 1, if it manages to read number, or `EOF`. The call to `printf()` will write the number to `stdout`.

```
/* ones.c */
#include <stdlib.h>
#include <stdio.h>

int main() {

  for(int n = 5; n > 0; n--) {
    printf("%d\n", n);
  }

  return 0;
}
```

```
/* add2.c */
#include <stdlib.h>
#include <stdio.h>

int main() {

  int in;
  int result = scanf("%d", &in);

  while(result != EOF) {
    printf("%d\n", in+2);
    result = scanf("%d", &in);
  }

  return 0;
}
```

You have a Linux computer and all possible programs. How would you in the simplest possible way make the output from the first program, `ones`, be read by the the other program `add2`.

Answer:

```
$> ./ones | ./add2
7
6
5
4
3
```

Name:____________________________ Persnr:____________________________

2.2 Shared memory [2 points*]

We can make two processes share memory by memory map a file in the two processes using `mmap`. This memory will then be visible to both processes and they can share it, almost as two threads can share the heap.

In the example below have the code to map a file that can then be used by several processes. We also have an extract from the man pages of `mmap()`.

```
int fd = open("shared", O_CREAT | O_RDWR, S_IRUSR|S_IWUSR);

// make sure the file is 4K byte
lseek(fd, 4096, SEEK_SET);
write(fd, "A", 1);

char *area = (char*)mmap(NULL, 4096, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
    :
    :
```

```
      void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);

DESCRIPTION
      mmap()  creates a new mapping in the virtual address space of the calling
      process.  The starting address for the new mapping is specified in  addr.
      The  length  argument  specifies the length of the mapping (which must be
      greater than 0).

      If addr is NULL, then the kernel chooses the  (page-aligned)  address  at
      which to create the mapping; this is the most portable method of creating
      a new mapping.  If addr is not NULL, then the kernel takes it as  a  hint
      about  where  to place the mapping; on Linux, the mapping will be created
      at a nearby page boundary.  The address of the new mapping is returned as
      the result of the call.
         :
         :
```

What would happen if we want to share linked data structures, what problem would we have and how could we handle it?

Answer: The problem is that the shared memory can be mapped to different virtual memory ranges. That to which one process looks like a valid reference is for the other process an address that is pointing somewhere completely different place. We can solve the problem by either making sure that the file is mapped to the same virtual addresses or encoding all references as offsets from the start of the area.

Name:____________________________ Persnr:____________________________

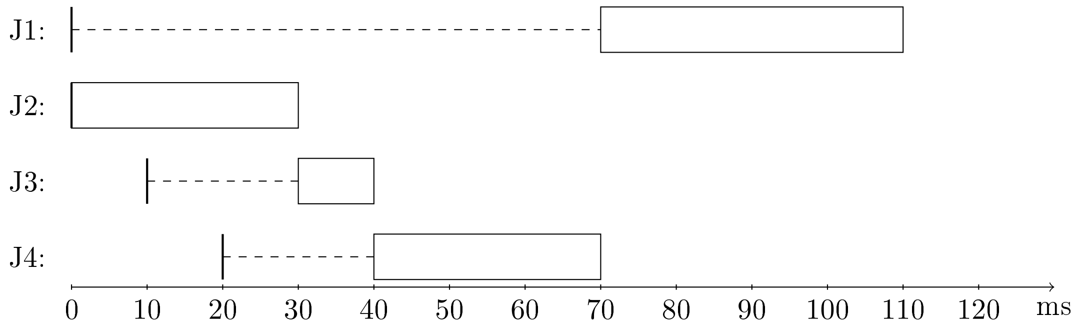
3 Scheduling

3.1 Bonnie Tylor [2 points]

Assume that we have a scheduler that implements *shortest job first*. We have four jobs described below as ⟨*arrive at*, *execution time* ⟩ in ms. Draw a time diagram and specify the turnaround time for each of the jobs.

Answer:

- J1 : ⟨0,40⟩ 110 ms
- J2 : ⟨0,30⟩ 30 ms
- J3 : ⟨10,10⟩ 30 ms
- J4 : ⟨20,30⟩ 50 ms



Name:____________________________ Persnr:____________________________

3.2 stride scheduling [2 points*]

One could implement a *stride scheduler* by keeping all processes in list sorted by *pass value*. The process that is first in the list is the one selected for execution. When the process has executed it is inserted in the list again, at what position should it be added.

Answer: Each process has a *stride value* that is added to its *pass value*. When a process is added to the list it will inserted at the position determined by its new pass value.

Name:____________________________ Persnr:____________________________

4 Virtual memory

4.1 you win some ,you loose some [2 points]

Assume that we have a paged virtual memory with a page size of 4Ki byte. Assume that each process has four segments (for example: code, data, stack, extra) and that these can be of arbitrary but given size. How much will the operating system loose in internal fragmentation?

Answer: Each segment will in average give rise to 2Ki byte of fragmentation. This will in average mean 8 Ki byte per process.

If we for example have 100 processes this is a total loss of 800 Ki byte.

4.2 paged memory with 64 byte pages [2 points*]

You have been asked to propose an architecture for a processor that should have a paged virtual memory with the page size as small as 64 byte. The processor is a 16 bit processor and the virtual address space should be 2^{16} bytes.

Propose a scheme that uses a hierarchical page table based on pages of 64 Ki byte and explain how the address translation is done.

Answer: One proposal is to use an offset of 6 bits and then have two levels in the tree with an index of 5 bits on each level. We need 6 bits as offset to address 64 bytes. The 5 bit used as index would mean that we could have 32 elements in a page table. If we encode each entry as two bytes we can encode a table in a page of 64 bytes. Using two bytes as an entry should be sufficient given, we could use 10 bits for a frame number and leave 6 bits for flags etc.

Name:____________________________ Persnr:____________________________

5 File systems and storage

5.1 what could happen [2 points]

Assume that we have simple file system without a journal where we write directly to bitmaps, inodes and data data blocks. Assume that we shall write to a file and that an additional data block is needed. When we perform the operations on disc, we only succeed in updating the inode but not the bit maps nor the selected data block before we crash.

If we do not detect the error when we restart, which problems will we have and what could happen?

Answer: We will have a data block that is allocated to an inode but the data block contains garbage and it is marked as free in the bit-maps. If we read from the file we will read garbage but worse if we use the data block for another file. This new file will then write its data to the block that can then be over written when we write to the first file. If the data block is used to represent a directory, this could of course result in total chaos.

Name:____________________________ Persnr:____________________________

5.2 log-based fs [2 points*]

In a log-based file system we write all changes in a continuous log without changing the existing data blocks that has been allocated to a file. We will sooner or later run out of blocks and need to reclaim blocks that are no longer used.

How do we keep track of which blocks that can be reused and what do we do to reclaim the blocks?

Answer: We need to identify the used block in the very back of the log. If we can move this block to the front of the log we can reused all consecutive blocks up to the next used block that is now the last used block in the log.

We maintain an inverse mapping that for a given block will tell us the inode of that the block belongs to. This means that we can determine if a block is used an if so, to which inode it belongs. If we copy the last block in the log to the front we also make a new copy of its inode with an updated sequence of blocks.