Reserved cells

Ex. 1	
Ex. 2	
Ex. 3	
Ex. 4	
Ex. 5	
Ex. 6	
Tot.	

Operationg Systems

Examination task

27 January 2020

ID number _____ Surname _____ Name _____

Professor: ◯ Scanzio

It is not possible to consult texts, notes or to use calculators. The only material allowed consists in the forms distributed by the professor. Solve the exercises in the reserved spaces. Additional sheets are permitted only when strictly necessary. Report the main steps for solving exercises.
Duration: 100 **minutes.**

1. Suppose to execute the following program

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
void *t1 (void *p){
  int *n = (int *) p;
  printf ("--- thread: id=%d\n", *n);
  pthread_exit (NULL);
}
int main () {
  pthread_t thread;
  int i, n, v[2];
  char str[100];
  setbuf (stdout, 0);
  for (i=0; i<2; i++)
    if (fork()>0) {
      v[i] = 0;
    } else {
      v[i] = 1;
    }
  n = v[0] + v[1]*2;
  pthread_create (&thread, NULL, t1, &n);
  pthread_join (thread, NULL);
  sprintf (str, "echo '- echo: n=%d'", n);
  system (str);
  sprintf (str, "-- exec: n=%d", n);
  execlp ("echo", "bash", str, NULL);
  return 1;
}
```

Report the *control flow graph* (CFG) and the *process generation tree* after its execution. Indicate what it produces on video and for what reason.

2. Describe the syntax of the system calls `wait` and `exit`, and an example on how to transfer with `exit` and `wait` an integer value from a terminated child to the parent. What happens if a parent does not call `wait` and a child terminates? Which mechanism is used by the *kernel* to identify the termination of a process?

   Two child processes, P_1 (with PID 123) and P_2 (with PID 456), which are the only children of a parent, terminates. Indicate for the following two codes the output provided by the function `printf` in the case P_1 terminates before P_2, and P_2 terminates before P_1 (4 cases in total).

Code 1:

```
while(wait()!=123);
pid = wait();
printf("%d\n", pid);
```

Code 2:

```
waitpid(123, (int *) 0, 0);
pid = wait();
printf("%d\n", pid);
```

3. Illustrate the "*Producer and Consumer*" problem, and report with pseudo-code a possible implementation schema. Indicate and motivate the function of all the semaphores.

   Then, adapt the previous solution to the case of exactly 3 producers and only one consumer. Each producer must generate elements in a queue dedicated to each process, the consumer must consume elements ensuring a higher priority to the queue with the major number of stored elements.

   Suggestion: use counters to track the number of elements in each queue, or alternatively use a function (e.g., `sem getvalue`) that can return the value of a semaphore.

4. Implement a BASH script that receives the path of a directory from command line. The script, after checking the passage of the correct number of parameters, it must select, in the sub-tree of directors with the specified directory as root, all the regular files with dimension less than 10MB, whose name starts with the string "expense" followed by an unsigned integer number and with extension `.xyz` (e.g., `expense1.xyz`, `expense200.xyz`).

   Assume that each of these files contains a text with a format similar to the following:

expense1.xyz

```
Product Quantity Unit_price
pasta 2 5
pizza 1 8
pasta 1 6
```

expense200.xyz

```
Product Quantity Unit_price
pizza 2 2
fruit 3 5
```

   where the first line is a header, while the following contain the name of a product, its quantity, and its unit price (separated by single spaces).

   For each selected file, the script must generate a file with the same name but with extension `.dat`, without header, that contains for each product the total, obtained by summing the quantities multiplied by the unit price of all the lines in which that product appears.

   For the example files shown above, the generated files must be the following:

expense1.dat

```
pasta 16
pizza 8
```

expense200.dat

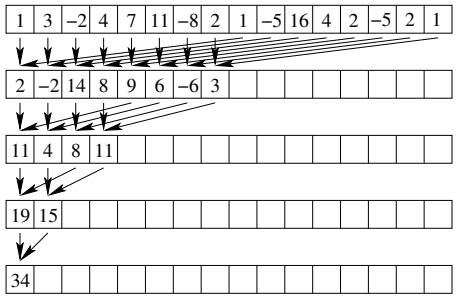
```
pizza 4
fruit 15
```

5. A function receives as parameters a vector of integers (`vet`) and its dimension (`n`), which is supposed to be equal to a power of 2:

   ```
   int array_sum (int *vet, int n);
   ```

   The function must return the sum of the elements of the vector. The sum has to be computed using a concurrent version of the following algorithm, which is illustrated in the figure for a vector with dimension `n = 16`:

   ```
   int i, k;
   k = n/2;
   while (k != 0) {
     for (i=0; i<k; i++) {
       vet[i] += vet[i+k];
     }
     k=k/2;
   }
   ```

   In particular, the function must apply the steps of the previous algorithm, ensuring that all sum operations are executed (in parallel) by $n/2$ separate threads. Each thread is associated with one of the first $n/2$ cells of the vector. Each thread takes care of executing all the sums whose result must be stored in the cell of the vector associated with it. Note that the number of sums each thread will have to execute depends on the position of the cells of the vector associated with it. Manage synchronization between threads with semaphores, so that all sums are made respecting precedences.

6. Clarify the main differences between an ASCII (or text) and a binary file. What advantages and disadvantages do the latter offer?

   Illustrate the main differences between the functions `fopen` and `open`, between `fprintf` and `write`, and between `fscanf` and `read`.

   Explain the differences between the *linked* and *indexed* allocations when saving files, illustrating their advantages and disadvantages. For the indexed allocation in the UNIX/LINUX environment, you have also to indicate what is the meaning of the terms “directory block”, “directory entry”, “data block” and “i-node”.