

Laboratorio di Sistemi Operativi – A.A. 2020/2021 – Totale e 1° Parziale del 28/01/2021

Cognome:	Nome:
Matricola:	Anno immatricolazione:
<input type="checkbox"/> 1° Parziale (40min) ESERCIZI dal 1 al 7	<input type="checkbox"/> Totale 6 CFU (1h) ESERCIZI dal 1 al 10
<p>PROPEDEUTICITÀ: lo studente è consapevole che NON PUÒ SOSTENERE questo esame SE NON SI È SUPERATO l'esame di "Laboratorio di Programmazione di Sistema - LPS" (precedentemente "Laboratorio di Architettura degli Elaboratori - LAE")</p> <p><input type="checkbox"/> Lo studente dichiara di aver già superato l'esame di LPS (LAE).</p> <p><input type="checkbox"/> Lo studente dichiara di voler sostenere questo esame CON RISERVA SUL SUPERAMENTO DI LPS IN UNO DEGLI APPELLI DELLA SESSIONE CORRENTE. Lo studente è consapevole che PERDERA' IL VOTO DI QUESTO ESAME SE NON SUPERERA' LPS IN UNO DEGLI APPELLI DELLA SESSIONE CORRENTE.</p>	

- Domande a risposta multipla → **2 punti** per ciascuna risposta esatta, **-1 punti** per ciascuna risposta errata, **0 punti** per ogni risposta omissa. Le domande a risposta multipla possono avere una e una sola risposta esatta.
- Per indicare la risposta scelta cambiare il **colore del testo in rosso**, oppure sottolineare, oppure **evidenziare**.

1. Indicare quale dei seguenti comandi consente di modificare i permessi del file "pippo" dalla maschera r-- -w- r-x alla maschera rw- rw- -w-

- a. `chmod u+w pippo; chmod a+x pippo`
- b. `chmod 642 pippo ; chmod g+w pippo`**
- c. `chmod 642 pippo ; chmod o+r pippo`

2. Il comando `file ./.*e* | while read A B; do echo $A $B ; done` stampa:

- a. il nome seguito dal tipo di tutti i file contenuti nelle directory i cui nomi iniziano con `"*e"`
- b. il solo tipo di tutti i file nella directory corrente i cui nomi iniziano con `"e"`**
- c. il nome seguito dal tipo di tutti i file nella directory corrente i cui nomi finiscono con `"e"`

3. Il comando `chmod -N *`

- a. stampa la lista di permessi Nulli (non validi) associati al file `<some file>`
- b. rimuove l'Access Control List associata a tutti i file nella directory corrente**
- c. rimuove l'Access Control List associata al file `<some file>`

4. Il comando `chmod -a# 1 "/deny/pluto/pippo"`

- a. cancella la entry ACL con indice 1 al file `./deny/pluto/pippo`**
- b. cambia la entry ACL per togliere i permessi di scrittura all'utente pluto e darli all'utente pippo
- c. aggiunge una entry ACL al file pippo per revocare il permesso di scrittura all'utente pluto

5. Scrivere nella sola riga in basso a destra cosa stampa il comando `es5.sh 5`

es5.sh	
<pre style="margin: 0;">#!/mybin/bash function myfunction { if [\$1 -gt 0]; then ((a=\$1-1)) temp1=\$(myfunction \$a) ((res=\$1*temp1)) else res=1 fi }</pre>	

<pre> echo \$res } res=\$(myfunction \$1) echo \$res </pre>	<u>120</u>
--------------------------------------------------------------	------------

6. Scrivere nella riga sotto un esempio di regola che un Makefile può contenere per evitare conflitti con un file avente lo stesso nome:

Rm -f obj/*.o *~ core ./include/*~

7. Sapendo che la funzione "preprocessing ()" cancella ogni occorrenza di "<" e sostituisce ogni occorrenza di ">" e "/" con ":" scrivere nelle tre righe sotto cosa stampa il comando ./es7.sh es7.xml

es7.sh	es7.xml
<pre> #!/mybin/bash xmlgetnext () { local IFS=';' read TAG VALUE TAG } pre_processing () { ... sed -e 's/</g' -e 's/>/:/g' -e 's/\/:/g' } cat \$1 pre_processing sed s/top/min/g while xmlgetnext ; do echo "\$TAG \$((++CONT)) :: \$VALUE" done </pre>	<pre> <data>topolino 4</data> <data>minnie 16</data> <data>pluto 2</data> </pre>

data 1 :: minolino 4

data 2 :: minnie 16

data3 :: pluto 2

8. Scrivere nelle due righe sotto, quali sono le sotto fasi della fase di compilazione C studiate in classe.

Parsing e analisi semantica

Generazione codice e ottimizzazione

9. Scrivere nelle righe sotto, come viene interpretato il valore del parametro *pid* dalla funzione *waitpid()*:

Se *pid* == 1 la funzione aspetta il figlio avente pid=1

Se *pid* > 0 la funzione aspetta il figlio avente process id uguale al valore di pid

Se *pid* == 0 la funzione aspetta qualsiasi figlio avente process group ID uguale a quello del processo chiamato

Se *pid* < -1 la funzione aspetta Qualsiasi cui process group id è uguale al valore assoluto di pid

10. Scrivere nelle righe sotto (non usare tutte le righe) cosa, molto probabilmente, stampa il seguente programma sapendo che alla fine (long)tret1 vale 21 e (long)tret2 vale 701408733 (N.B.: oltre 700 milioni!)

```
#include "apue.h"
#include <pthread.h>

long job(int i)
{
    if (i < 0) return -1;

    if (i == 0) return 0;
    else if (i == 1) return 1;
    else return job(i-1) + job(i-2);
}

void *
thr_job(void *arg)
{
    long res = job((int)arg);
    printf("Job done! res: %ld\n", res);
    return((void *)res);
}

int
main(void)
{
    int err;
    pthread_t tid1, tid2;
    void *tret1,*tret2;

    err = pthread_create(&tid1, NULL, thr_job, (void *)8);
    if (err != 0)
        err_exit(err, "can't create thread 1");

    err = pthread_create(&tid2, NULL, thr_job, (void *)44);
    if (err != 0)
        err_exit(err, "can't create thread 2");

    err = pthread_join(tid1, &tret1);
    if (err != 0)
        err_exit(err, "can't join with thread 1");
    printf("First thread joined!\n");

    err = pthread_join(tid2, &tret2);
    if (err != 0)
        err_exit(err, "can't join with thread 2");
    printf("Second thread joined!\n");

    printf("Final result %ld\n", (long)tret1 + (long)tret2);

    exit(0);
}
```

Job done! res: 21
First thread joined!
Job done! res: 701408733
Second thread joined!
Final result 701408754