CALIFORNIA STATE UNIVERSITY, LONG BEACH

College of Engineering
Department of Computer Engineering and Computer Science
Mr. Haney Williams
CECS-326: Operating Systems
PROJECT 1
BINARY TREE OF PROCESSES

Name: Pan Amanda
Last, First

ABSOLUTELY NO LATE PROJECTS WILL BE ACCEPTED

PROJECT DESCRIPTION:

Write a program using C/C++ to generate a binary tree of processes. The input to the program includes the number of levels in the tree. The maximum number of levels is 5, but the program should work for a general case of any number. Use command line arguments.

Example of a program run is as follows: **ecs416lin213.cecs.csuld.edu:1>p1 4**

Level No.	Procs ID	Parent ID	Child 1 ID	Child 2 ID
0	3608	3574	3609	3610
1	3610	2608	3612	3613
1	3609	3608	3611	3614
2	3612	3610	3616	3617
2	3614	3609	3620	3621
2	3611	3609	3615	3622
2	3613	3610	3618	3619
3	3620	3614	3629	3630
3	3622	3611	3633	3634
3	3615	3611	3623	3635
3	3616	3612	3624	3625
3	3617	3612	3626	3636
3	3618	3613	3627	3637
3	3619	3613	3628	3638
3	3621	3614	3631	3632

DOCUMENTATION:

The project must be carefully documented. The write-up must include: one page of problem analysis and discussion (include a flow chart or structure chart when relevant), program listing with proper comments, one page of output listing using N = 3. Draw a tree diagram showing the process IDs in circles. The discussion should include general considerations, UNIX/LINUX function calls, any reasonable assumptions, and any other relevant information. Attach this sheet as the front page of your project. Firmly staple the entire project together. No folder is used.

NOTES:

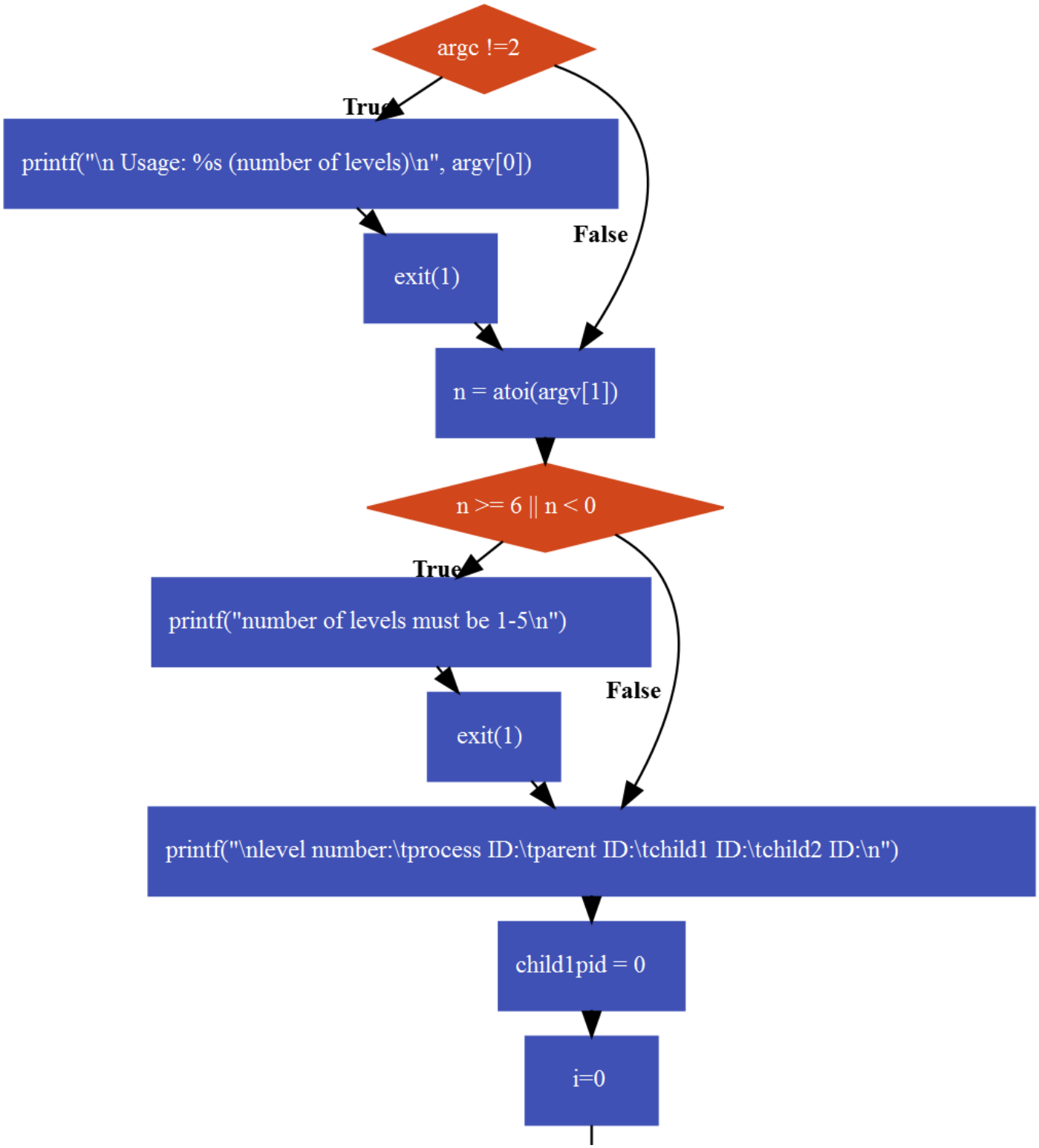
1) After submitting the project, do not change, edit, or recompile the program. Violations of this rule will result in the project being graded zero.
2) Discussions among students are encouraged, but only to the extent of clarifying the problem statement and/or the general approach to the problem as discussed in class. The project must represent your own work.

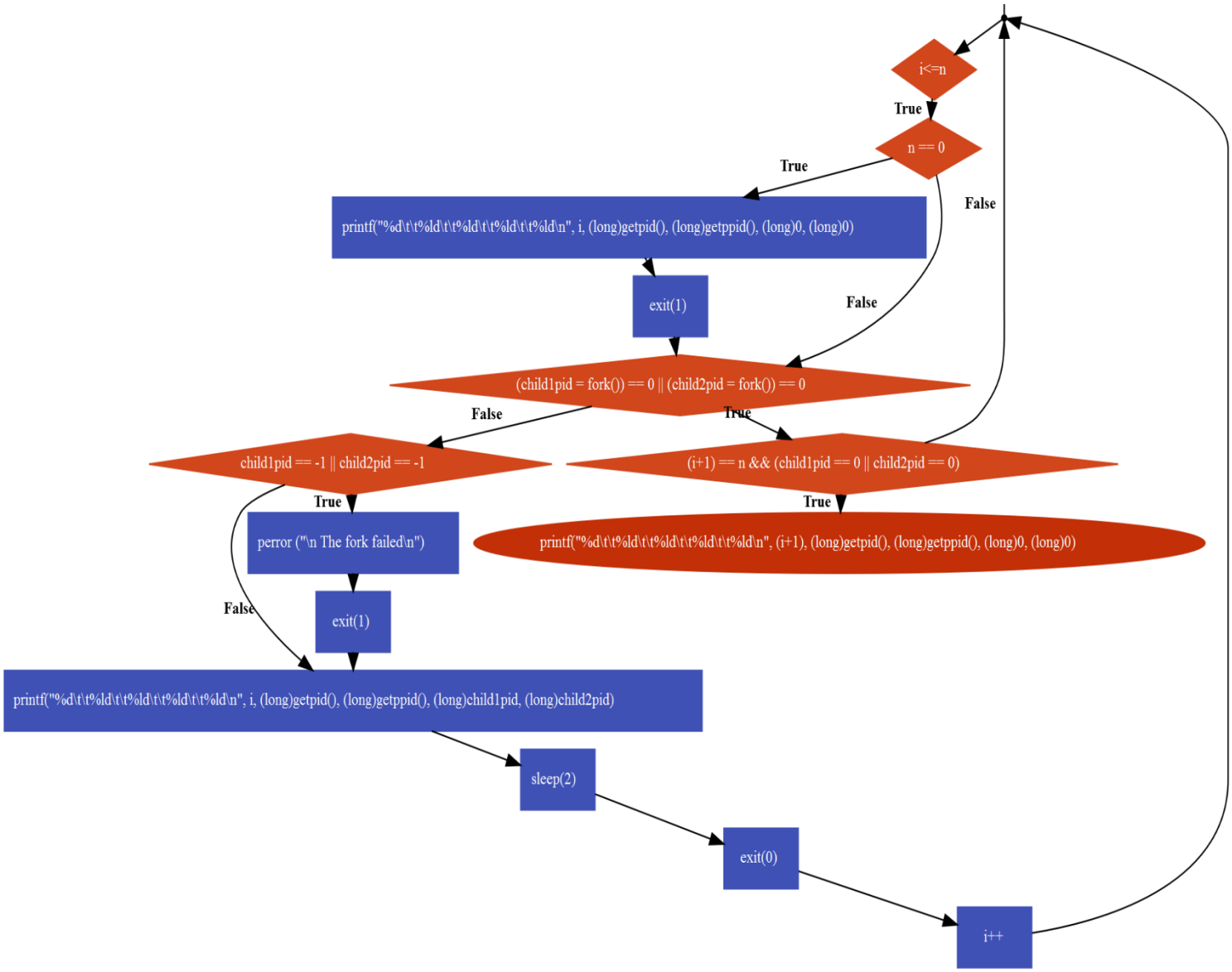
END OF PROJECT 1

Problem Analysis:

The problem uses only *proj1* program. The program generates a binary tree of a maximum of 5 levels, depending on the input of the user. The process creates two child process using *fork()*.

Discussion:




i<=n
True
n == 0
True
False
printf("%d\t\t%ld\t\t%ld\t\t%ld\t\t%ld\n", i, (long)getpid(), (long)getppid(), (long)0, (long)0)
exit(1)
False
(child1pid = fork()) == 0 || (child2pid = fork()) == 0
False
True
child1pid == -1 || child2pid == -1
(i+1) == n && (child1pid == 0 || child2pid == 0)
True
True
perror ("\n The fork failed\n")
printf("%d\t\t%ld\t\t%ld\t\t%ld\t\t%ld\n", (i+1), (long)getpid(), (long)getppid(), (long)0, (long)0)
False
exit(1)
printf("%d\t\t%ld\t\t%ld\t\t%ld\t\t%ld\n", i, (long)getpid(), (long)getppid(), (long)child1pid, (long)child2pid)
sleep(2)
exit(0)
i++

Programing Listing:

```
/*****************************************************************************/
/* PROGRAM: proj1.c */
/* DESCRIPTION: This program generates a binary tree */
/* using fork(). The number of levels n is a command line argument. */
/* Each process sleeps for 2 seconds then prints out */
/* Level number, process ID, parent ID, child1 ID, and child2 ID */
/*****************************************************************************/
# include <stdio.h>
# include <stdlib.h>
# include <sys/types.h>
# include <unistd.h>
# include <sys/wait.h>
# include <signal.h>

int main(int argc, char *argv[])
{
    int i, n;
    pid_t child1pid, child2pid;
    //if argument entered is not 2, prompt error and exit
    if (argc !=2) {
        printf("\n Usage: %s (number of levels)\n", argv[0]);
        exit(1);
    }
    n = atoi(argv[1]); //n takes in the second argument
    //value of n must be between 1-5, or else exit
    if (n >= 6 || n < 0) {
        printf("number of levels must be 1-5\n");
        exit(1);
    }
    printf("\nlevel number:\tprocess ID:\tparent ID:\tchild1 ID:\tchild2
ID:\n");
    child1pid = 0;
    for (i=0; i<=n; i++) {
        if (n == 0)
        {
            printf("%d\t\t%ld\t\t%ld\t\t%ld\t\t%ld\n", i, (long)getpid(),
(long)getppid(), (long)0, (long)0);
            exit(1);
        }
        //if either child pid is 0, continue the loop
        if ((child1pid = fork()) == 0 || (child2pid = fork()) == 0) {
            //if child pid is the last level, print to show that it is a leaf
            if((i+1) == n && (child1pid == 0 || child2pid == 0))
            {
                return printf("%d\t\t%ld\t\t%ld\t\t%ld\t\t%ld\n", (i+1),
(long)getpid(), (long)getppid(), (long)0, (long)0);
            }
        continue;
        }
        //if child pid is 1-, fork failed and exit
        if (child1pid == -1 || child2pid == -1) {
```

```
            perror ("\n The fork failed\n");
            exit(1);
        }
        printf("%d\t\t%ld\t\t%ld\t\t%ld\t\t%ld\n", i, (long)getpid(),
(long)getppid(), (long)child1pid, (long)child2pid);
        sleep(2); //process sleeps for 2 seconds
        exit(0);
    }
}
```

Output Program:

```
level number:   process ID:     parent ID:      child1 ID:      child2 ID:
0               2308            2151            2309            2310
1               2310            2308            2311            2313
2               2312            2309            2315            2317
2               2314            2309            2316            2318
2               2313            2310            2319            2320
1               2309            2308            2312            2314
2               2311            2310            2321            2322
3               2318            2314            0               0
3               2315            2312            0               0
3               2320            2313            0               0
3               2322            2311            0               0
3               2316            2314            0               0
3               2319            2313            0               0
3               2317            2312            0               0
3               2321            2311            0               0
```

Tree diagram: