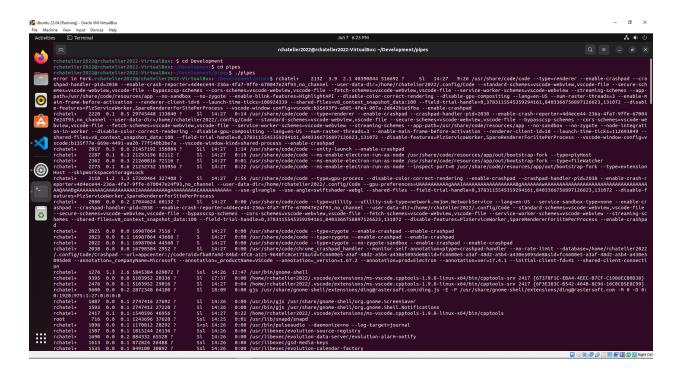
Problem 1

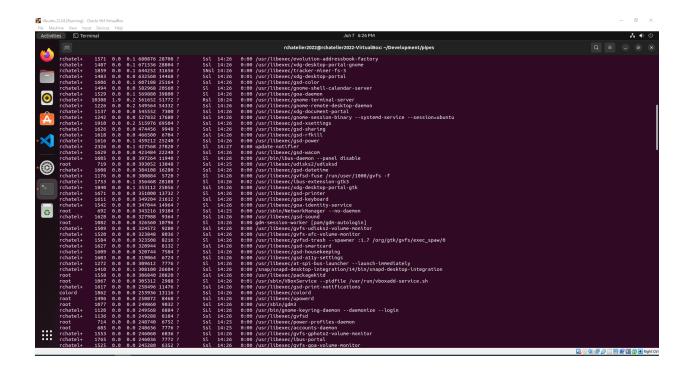
```
#include <unistd.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char* argv[]) {
  int pipes[2];
  int idpipe;
  if (pipe(pipes) < 0) {</pre>
      printf("Pipe doesn't work\n");
  idpipe = fork();
  if (idpipe == 0) {
      dup2(pipes[1], 1);
      close(pipes[0]);
      execlp("ps", "ps", "-aux", (char *)NULL);
      int idpipe = fork();
       if(idpipe == 0) {
          close(pipes[1]);
          dup2(pipes[0], 0);
           execlp("sort", "sort", "-r", "-n", "-k", "5", (char *)NULL);
```

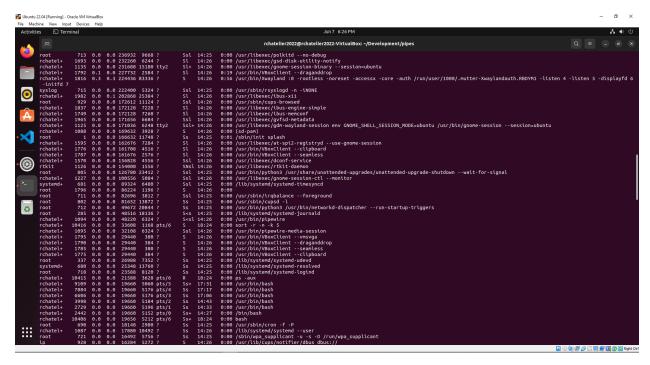
```
else {
    printf("error in Fork.");
    return 3;
}

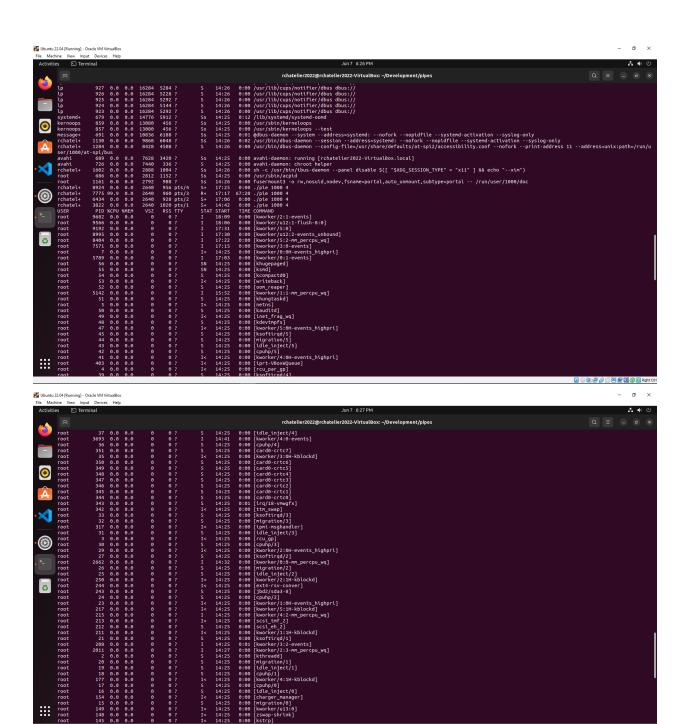
// pipes closed
    close(pipes[1]);
    close(pipes[0]);
    wait(NULL);
}

return 0;
}
```

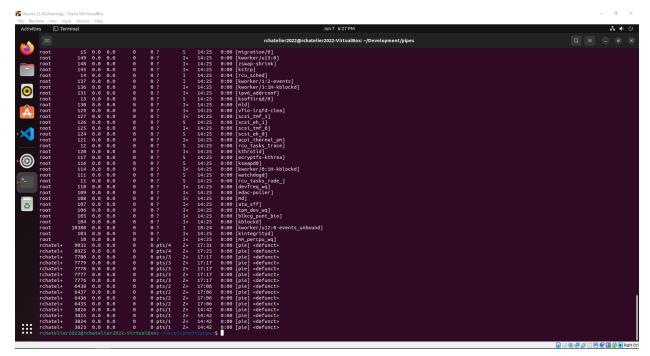








:::



Displaying all of the auxiliary processes, which are sorted by the k5, r, and n.

Problem 2

```
#include <unistd.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
   int pipe1[1000][2];
   int pipe2[1000][2];
   int n = atoi(argv[1]);
   int t = atoi(argv[2]);
   double collect = 3;

// t and n compared in boolean expression
   if (t > n)
   {
      // check if error of N is greater than T
      printf("Error N is greater than T\n");
```

```
exit(1);
int number1 = pipe(pipe1[j]);
int number2 = pipe(pipe2[j]);
if (number1 < 0)</pre>
    printf("Pipe 1 has failed.\n");
    exit(1);
if (number2 < 0)
    printf("Pipe 1 has failed.\n");
    exit(-1);
pid t idpipe = fork();
if (idpipe == 0)
    double computePartialSum(int n, int t, double j);
    int b;
    read(pipe1[j][0], &a, sizeof(int));
    read(pipe1[j][0], &b, sizeof(int));
    read(pipe1[j][0], &c, sizeof(int));
```

```
close(pipe1[j][1]);
           double slice = computePartialSum(a, b, c);
           write(pipe2[j][1], &slice, sizeof(double));
           close(pipe2[j][1]);
          printf("\n");
          exit(0);
          close(pipe1[j][0]);
          write(pipe1[j][1], &n, sizeof(int));
          write(pipe1[j][1], &t, sizeof(int));
          write(pipe1[j][1], &j, sizeof(int));
          close(pipe1[j][1]);
       read(pipe2[j][0], &b, sizeof(double));
      collect += b;
  while (wait(NULL) >= 0);
  printf("%1.02f\n", collect);
double computePartialSum(int n, int t, double j)
  double total = 0;
```

```
int compute1 = ((n * j) / (t)) + 1;
int compute2 = (n / t) * (j + 1);

for (int i = compute1; i <= compute2; i++)
{
      // checking for evens
      if (i % 2 == 0)
      {
            q = -1;
      }
      // checking for odds
      else
      {
            q = 1;
      }
      // Nilakantha approximation formula used to find pi
      total += q * (double)(4) / ((2 * i) * (2 * i + 1) * (2 * i + 2));
}

return total;
}</pre>
```

