Team member : Zhen Shi
Jin Jin

1. General

Aim
Read temperature values from sensors to 20 sensor descriptors. After that read all 20 sensor descriptors at same time. Deal with the data reading from sensor descriptors and calculate the maximum, minimum and average delay.

Project Plan
Step 1
According to the project requirement, trying to read 5 temperature values from sensor to sensor descriptors and printing 1 measurement values and time values from the first file descriptor
Step 2
Print delay time between two time values from one file descriptor.
Step 3
Build structure for reading twenty file descriptors.
Step 4
Merge step 2 and 3. Read data in different processes.
Step 5
In order to get the total sum of delay time, we have to let parent process and child process communicate.
Step 6
Finally, compare all the values and calculate the max, mini and average delay time.

Programming language: C programing language.

Team division
We decided to do the pair programming and try to finish it in 3 days (3 hours per day). There is no particular task division, we will work like one person writing the code for 1.5 hours, another sitting beside him navigate and supervise. After 1.5 hours the role exchanges.

2. Function specification

The system supposed to read the temperature value from 20 temperature sensors and put them into sensor file descriptors. The program can read all values from file descriptors simultaneously. After reading all temperature values the system will calculate the sum of delay time, average delay time, minimum delay and maximum delay time among all the sensor file descriptors.

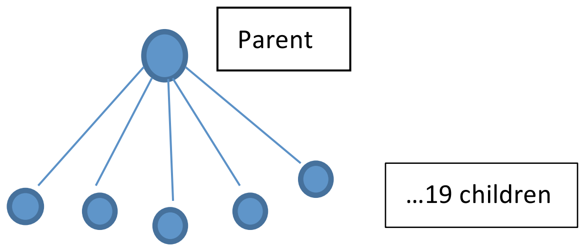
3. Technical specification

There are four different solutions for reading twenty file descriptors in real-time: separate processes, multiplexed i/o, asynchronous i/o and threads. We chose to use separate processes to archive this project. Since multi-processes is most familiar for us, we make an agreement of implementing the function with it.

In order to read all values simultaneously, the system have to build 20 processes and each processes read one file descriptor and deal with this file descriptor so that 20 file descriptors can be read at same time.

For the separate processes, it is easy to build several child processes by using system call fork(). However the structure of separate processes should be one parent with many children. In this case, should be one parent waiting for 19 children back at same time read values from file descriptors. The relation graphic should like following graphic.

.



Firstly, We encapsulated the given interface function into one self-defined function "start_read" to read from one file descriptor, so that we can reuse it and easily read data in multi-processes.

Then we used a certain pattern for creating a certain number of child processes we want, which forks in parent process. We started with making a testing program for printing a sequenced number in every process. After creating a certain number of child, reuse the function "start_read" in twenty processes for reading.

After that, we build up a pipe for twenty children to write and parent to read. We found out we do not need non-block piping since there's only one reading end and the pipe size is big enough for our case.

Finally, we use bubble sorting to calculate the max and min time delay among the 19 child processes. Then compare them with those of parent process.

4.Detailed documentation

```
void initial_time(struct timespec* time)
{
    time->tv_sec=0;
    time->tv_nsec=0;
}
```

List 1. Initial time function

This function is for time initialize. The time value is used for the parameter to this function.

```
struct timespec start_read(int* sensorDescriptor){
    Tmeas measurement[NUMOFVALUE];
    struct timespec diff,sum;
    int i;
    initial_time(&sum);
    for(i=0;i<NUMOFVALUE;i++){
        read(*sensorDescriptor, &measurement[i], sizeof(Tmeas));
        printf("Measurement value was %d\n", measurement[i].value);
        if(i>0){
            diff=diff_timespec(&measurement[i-1].moment,&measurement[i].moment);
            increment_timespec(&sum,&diff);
                }

    }
    printf("pid: %d sum value was %lld.%.9ld\n",getpid(),sum.tv_sec,sum.tv_nsec);
    return sum;
}
```

List 2.Read time

This function (start_read(int* sensorDescriptor) is use to read temperature value from sensor decirptors and print the sum of delay time from this file descriptor.

```
int main(void)
{
    int noOfSensors,i=0;
    pid_t pid;
    int sensorDescriptor [20];
    int status;
    int pipe_arr[2];
    struct timespec return_csum,total_sum,return_psum,csum_max,csum_min;
    initial_time(&total_sum);
    initial_time(&return_csum);
    initial_time(&csum_max);
    initial_time(&csum_min);
```

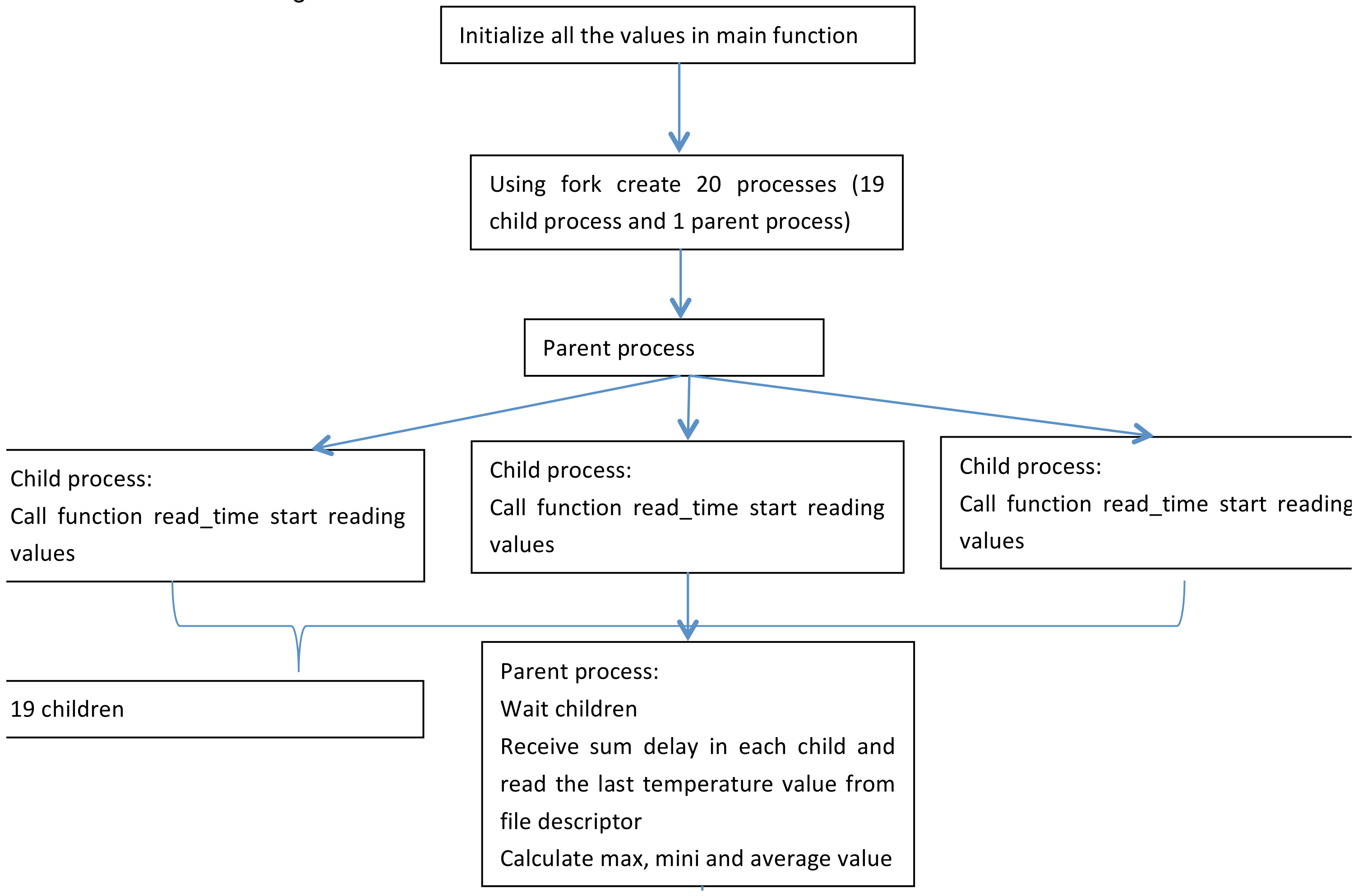
```
pipe(pipe_arr);


noOfSensors = StartSimulator(sensorDescriptor, NUMOFVALUE);

while (i < NUMOFCHILDREN) {

pid = fork();


if (pid == 0) { // this is child
     // this is child.
     // tasks of child process are performed.
    close(pipe_arr[0]);
    return_csum=start_read(&sensorDescriptor[i]);
    write(pipe_arr[1],&return_csum,sizeof(return_csum));

    exit(0);
   // child terminates.
     }
     i++;


}
// Parent continues from here after creating
// all children.
// Tasks of parent process    are performed.
//sum up children
close(pipe_arr[1]);

while(read(pipe_arr[0],&return_csum,sizeof(return_csum))>0){
    increment_timespec(&total_sum,&return_csum);
    //bubble compare
    //calculate csum_max
    if (return_csum.tv_sec>csum_max.tv_sec)
        csum_max=return_csum;
    else if (return_csum.tv_sec==csum_max.tv_sec)
    {
        if (return_csum.tv_nsec>csum_max.tv_nsec)
            csum_max=return_csum;
    }
    //initial min
    if (csum_min.tv_sec==0&&csum_min.tv_nsec==0)
    {
```

```
                csum_min=return_csum;
            }
            //calculate csum_min
            if (return_csum.tv_sec<csum_min.tv_sec)
                csum_min=return_csum;
            else if (return_csum.tv_sec==csum_min.tv_sec)
            {
                if (return_csum.tv_nsec<csum_min.tv_nsec)
                    csum_min=return_csum;
            }
        }

        //sum up with parent
        return_psum=start_read(&sensorDescriptor[i]);
        increment_timespec(&total_sum,&return_psum);

        //print total

        //compare child and parent for max
        printf("\n\n");
        if (csum_max.tv_sec>return_psum.tv_sec)
        {
            printf("max    value was %lld.%.9ld\n",csum_max.tv_sec,csum_max.tv_nsec);
        }
        else if(csum_max.tv_sec==return_psum.tv_sec){
            if (csum_max.tv_nsec>return_psum.tv_nsec)
                printf("max    value was %lld.%.9ld\n",csum_max.tv_sec,csum_max.tv_nsec);
            else
                printf("max                                              value
was %lld.%.9ld\n",return_psum.tv_sec,return_psum.tv_nsec);
        }else
            printf("max    value was %lld.%.9ld\n",return_psum.tv_sec,return_psum.tv_nsec);
        //compare child and paret for min
        if (csum_min.tv_sec<return_psum.tv_sec)
        {
            printf("min    value was %lld.%.9ld\n",csum_min.tv_sec,csum_min.tv_nsec);
        }
        else if(csum_min.tv_sec==return_psum.tv_sec){
            if (csum_min.tv_nsec<return_psum.tv_nsec)
                printf("min    value was %lld.%.9ld\n",csum_min.tv_sec,csum_min.tv_nsec);
            else
                printf("min                                              value
was %lld.%.9ld\n",return_psum.tv_sec,return_psum.tv_nsec);
        }else
        printf("min   value was %lld.%.9ld\n",return_psum.tv_sec,return_psum.tv_nsec);


     printf("total      time      value      was      %lld.%.9ld            average      delay
is  %lld.%.9ld\n",total_sum.tv_sec,total_sum.tv_nsec,total_sum.tv_sec/(NUMOFCHILDREN+
1),total_sum.tv_nsec/(NUMOFCHILDREN+1) );


     while(wait(NULL) > 0);
     exit(0);

     return 0;
}
```

List 3. Main function

Function first using initialize function to initialize the time value (maximum, minimum and sum) and then using read function read temperature value from sensor descriptor and all the tasks working in different processes which create by system call fork().After reading value, children have to communicate with their parent which send the sum of delay to their parent. In this case we use pipe to let them communicate. Finally, after received all the temperature values in the parent we compare them and find maximum value, minimum value and average of whole values.

flow chart diagrams


Initialize all the values in main function
Using fork create 20 processes (19 child process and 1 parent process)
Parent process
Child process: Call function read_time start reading values
Child process: Call function read_time start reading values
Child process: Call function read_time start reading values
19 children
Parent process: Wait children Receive sum delay in each child and read the last temperature value from file descriptor Calculate max, mini and average value