Fundamentos de Sistemas de Operação

MIEI 2018/2019

Laboratory session 10

Objectives

Processes communicating with pipes.

Counting numbers in an array

Remember the program from laboratory session 9. Assume now that you are programing for a system without Posix threads but assume that shared memory can be used among processes. For our example, in the following code, we are not using explicit shared memory for our array (as we should), but are assuming that after fork, the OS memory manager will share pages among processes using copy-on-write. As each new process just reads the array, we expect a limited use of extra physical memory by each one.

Your program, that counts the number of times a specified number appears in an array (vector) of integers, must be rewritten to make use of processes instead of Pthreads. Each child process works on an array slice and the result must be sent to the father process via a pipe. You can start from the following code:

```
int *array;  //data to work on (we are expecting that the OS will
             // use shared pages among all child processes)

int docount( int n, int size ){
    int count = 0;
    int start=n*size;
    int end=(n+1)*size;
    for (int i=start; i < end; i++) {
        if(array[i] == tofind)
            count++;
    }
    return count;
}

int launchWorker( int n, int size ) { // returns the pipe stream to read result from
    // TODO
}

int main(int argc, char const *argv[])
{
    struct timeval t1,t2;
    int size, count = 0;

    if ( argc!=2 ) {
        fprintf(stderr,"usage: %s num_procs\n", argv[0]);
        return 1;
    }
    int nprocs = atoi( argv[1] );
    printf("using %d procs\n", nprocs);
    array = (int *)malloc(SIZE*sizeof(int)); // global array
    tofind = 3;
    size = SIZE/nprocs;

    srand(0);
    for (int i=0; i < SIZE; i++) {
        array[i] = rand() % 4;
      }
    int *pipes = alloca(nprocs*sizeof(int));  // local array

    gettimeofday(&t1, NULL);
    for ( int p=0; p<nprocs && pipe[p-1]!=-1; p++ ) { // start workers
        pipes[p] = launchWorker( p, size );
    }
    for ( int p=0; p<nprocs; p++ ) {
        int c;
             // TODO: read results from all workers
        count += c;
    }
    gettimeofday(&t2, NULL);
```

```
    for ( int p=0; p<nprocs; p++ )
        wait(NULL) );

    printf("Count of %d = %d\n", tofind, count);
    printf("Elapsed time (ms) = %lf\n",
       ((t2.tv_sec - t1.tv_sec)*1000000 + (t2.tv_usec - t1.tv_usec))/1000.0 );
    return 0;
}
```

Do you need mutexes or other synchronization mechanism?

Measure the time that this sequential program takes to count number 3 in the array and compare with the best result from last laboratory session. Explain the execution time differences between all program versions.