```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <sys/time.h>
#include <sys/types.h>
#include <wait.h>

// errno is an external global variable that contains
// error information
extern int errno;

// get_seconds returns the number of seconds since the
// beginning of the day, with microsecond precision
double get_seconds () {
  struct timeval tv[1];

  gettimeofday (tv, NULL);
  return tv->tv_sec + tv->tv_usec / 1e6;
}


void child_code(int i)
{
  sleep (5);
  printf ("Hello from child %d.\n", i);
  exit (i);
}

// main takes two parameters: argc is the number of command-line
// arguments; argv is an array of strings containing the command
// line arguments
int main (int argc, char *argv[])
{
  int status;
  pid_t pid;
  double start, stop;
  int i, num_children;

  // the first command-line argument is the name of the executable.
  // if there is a second, it is the number of children to create.
  if (argc == 2) {
    num_children = atoi (argv[1]);
  } else {
    num_children = 1;
  }

  // get the start time
  start = get_seconds ();

  for (i=0; i<num_children; i++) {

    // create a child process
    printf ("Creating child %d.\n", i);
    pid = fork ();

    /* check for an error */
    if (pid == -1) {
      fprintf (stderr, "fork failed: %s\n", strerror(errno));
      perror (argv[0]);
      exit (1);
    }

    /* see if we're the parent or the child */
    if (pid == 0) {
      child_code(i);
    }
  }

  /* parent continues */
  printf ("Hello from the parent.\n");

  for (i=0; i<num_children; i++) {
    pid = wait (&status);

    if (pid == -1) {
      fprintf (stderr, "wait failed: %s\n", strerror(errno));
      perror (argv[0]);
      exit (1);
    }

    // check the exit status of the child
    status = WEXITSTATUS (status);
    printf ("Child %d exited with error code %d.\n", pid, status);
  }
  // compute the elapsed time
  stop = get_seconds ();
  printf ("Elapsed time = %f seconds.\n", stop - start);

  exit (0);
}
```