Assignment #1

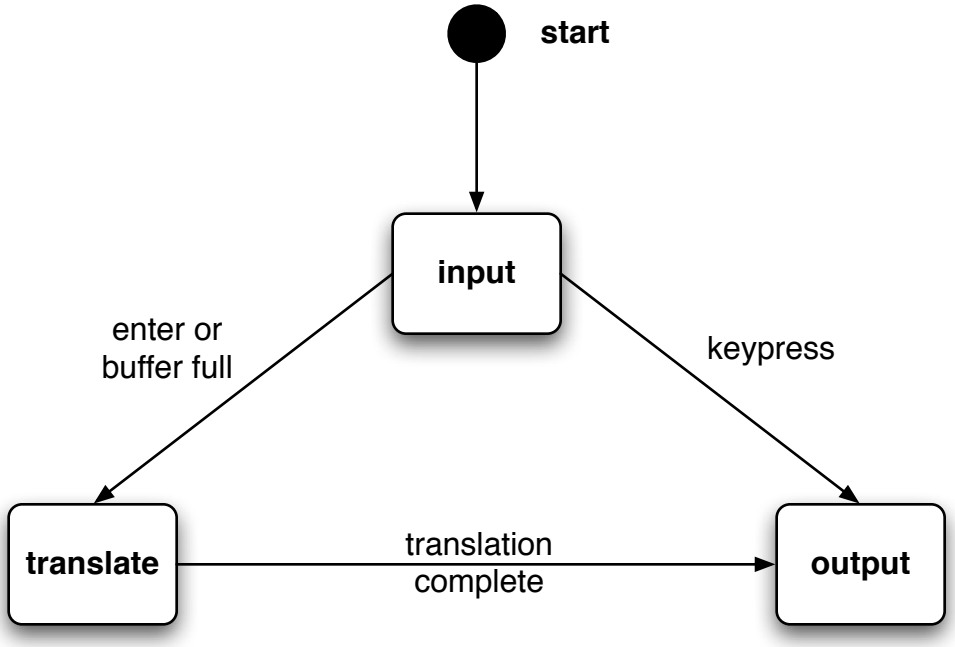
KeyCatcher

Steffen L. Norgren
A00683006

COMP 4981 • BCIT • January 22, 2009

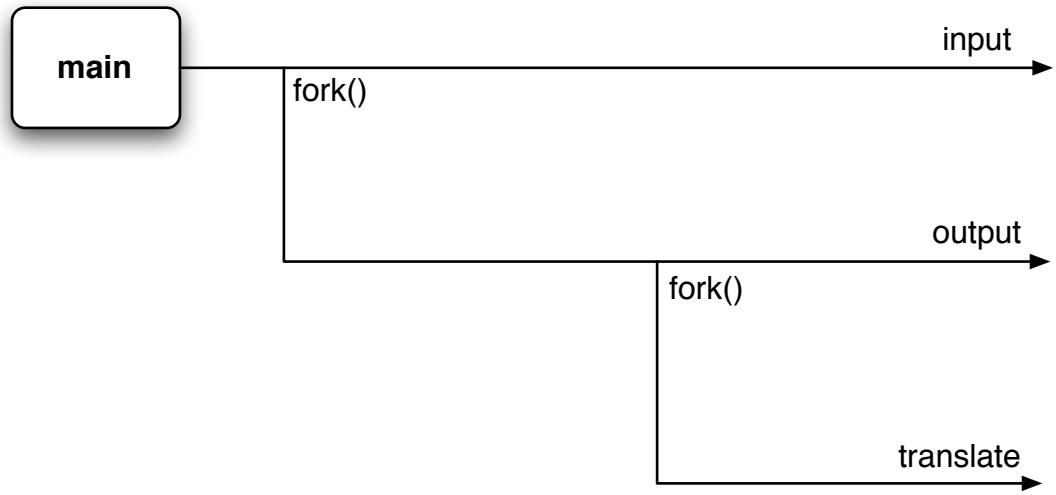
DESIGN

General Overview

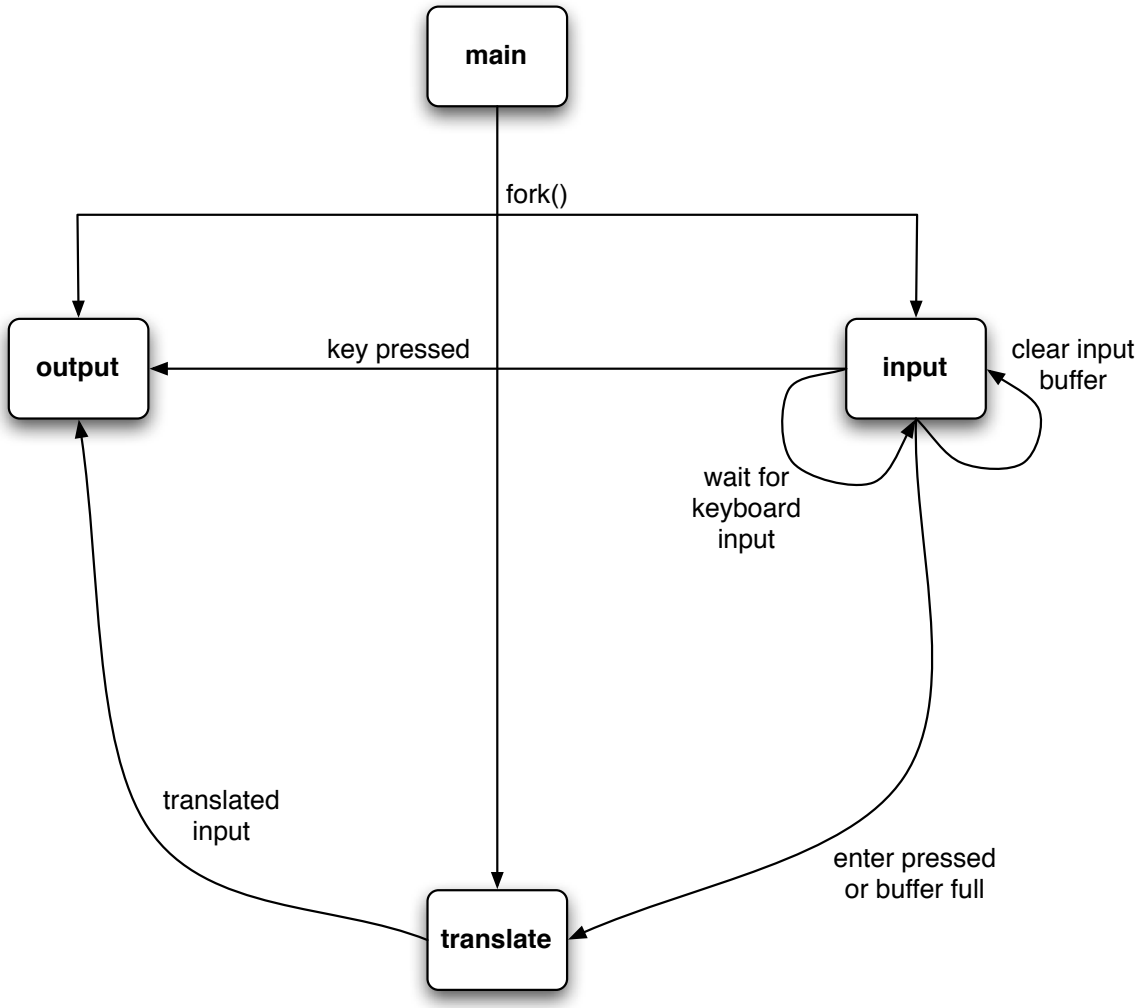


In the above diagram, input, output, and translate represent individual processes. The input process is the parent process, which forks to create the output and translate processes.

Each of these processes communicate with each other using pipes, which is represented by the arrowed lines connecting each process.



To create the three processes, the main process will fork twice, allowing for the child processes to handle output and translation functions. The parent process will remain responsible for input.



Main - Pseudo-Code

```
// Create pipes
int p1[2];
int p2[2];

pipe(p1);
pipe(p2);

switch (fork()) {
    case -1:
            // error

    case 0: // child 1
            output();
            break;
```

```
    default:
            switch(fork()) {
                  case -1:
                        //error

                  case 0: // child 2
                        translate();
                        break;

                  default: // parent
                        input();
                        break;
            }
}
```

Input - Pseudo-Code

```
while (c = getchar()) {
    if (c == enter) {
            buffer += c;
            write(p1[0], buffer, bufffer.length); // write to pipe
    }
    else if (c == backspace) {
            // ignore this key
    }
    else if (c == escape) {
            // ignore this key
    }
    else if (c == ctrl+k) {
            // cause abnormal termination
    }
    else {
            buffer += c;
    }
}
```

Output - Pseudo-Code

```
while (true) {
    if (read(p1[0], outbufff, outbuff.length) > 0) {
            printf("%s\n", outbuff);
    }
}
```

Translate - Pseudo-Code

```
while (true) {
    if (read(p2[0], outbufff, outbuff.length) > 0) {
            for (int i = 0; i < buffer.length; i++) {
                  if (buffer[i] == 'X') {
                        // remove the X and the previous char
                  }
                  else if (buffer[i] == 'K') {
                        // remove all previous characters
                  }
                  else if (buffer[i] == 'a') {
                        buffer[i] = 'z';
                  }
                  else if (buffer[i] == 'T') {
                        // signal normal termination
                  }
            }
            write(p2[1], outbuff, outbuff.length); // send to output
    }
}
```

TESTING

Tests & Results

The majority of my testing involved typing in various string combinations to verify that they conformed to the requirements. For example, the following test strings are examples of the types of tests I performed with regards to the translation function:

input: 123XXX456
output: 456

input: 123XXXXX456
output: 456

input: 123K456
output: 456

input: abcABC
output: zbcABC

Additionally, I verified that when the input buffer (80 characters) was reached, it would automatically send the buffer to the translation function for processing and display the results. Also along these lines, I attempted to copy and paste very large strings to see if the output would be fouled up in any way.

With regards to parent & child processes, I always had a secondary terminal window open running “watch -n 0.5 ps au” to verify that three processes were running during my testing. This allowed me to verify that all off the processes would exit when the program was terminated, leaving no zombie processes behind.