~\Desktop\ING DEL SOFTWARE\SOFTWARE 2_2\SISTEMAS OPERATIVOS\Prácticas\Práctica 4\prShellAmpliaciones\job_control.h

```
/*-----------------------------------------------------------
UNIX Shell Project
function prototypes, macros and type declarations for job_control module

Sistemas Operativos
Grados I. Informatica, Computadores & Software
Dept. Arquitectura de Computadores - UMA

Some code adapted from "Fundamentos de Sistemas Operativos", Silberschatz et al.
-----------------------------------------------------------*/

#ifndef _JOB_CONTROL_H
#define _JOB_CONTROL_H

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <termios.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>

// ----------- ENUMERATIONS ------------------------------------------
enum status { SUSPENDED, SIGNALED, EXITED, CONTINUED};
enum job_state { FOREGROUND, BACKGROUND, STOPPED, RESPAWNABLE };
static char* status_strings[] = { "Suspended", "Signaled", "Exited", "Continued"};
static char* state_strings[] = { "Foreground", "Background", "Stopped", "Respawnable" };

// ----------- JOB TYPE FOR JOB LIST ---------------------------------
typedef struct job_
{
    pid_t pgid; /* group id = process lider id */
    char * command; /* program name */
    enum job_state state;
    struct job_ *next; /* next job in the list */
    char ** args;   // Amp 1
} job;

// ----------- TYPE FOR JOB LIST ITERATOR ----------------------------
typedef job * job_iterator;

// -----------------------------------------------------------------------
//      PUBLIC FUNCTIONS
// -----------------------------------------------------------------------

void get_command(char inputBuffer[], int size, char *args[],int *background, int
*respawnable);

job * new_job(pid_t pid, const char * command, enum job_state state);

void add_job(job * list, job * item);

void add_respawnable_job(job * list, job * item, char ** args); // Amp 1

int delete_job(job * list, job * item);
```

```
55
56  job * get_item_bypid(job * list, pid_t pid);
57
58  job * get_item_bypos(job * list, int n);
59
60  enum status analyze_status(int status, int *info);
61
62  // -----------------------------------------------------------------------
63  //      PRIVATE FUNCTIONS: BETTER USED THROUGH MACROS BELOW
64  // -----------------------------------------------------------------------
65
66  void print_item(job * item);
67
68  void print_list(job * list, void (*print)(job *));
69  void print_background_list(job * list, void (*print)(job *));   // Amp extra
70  void print_stopped_list(job * list, void (*print)(job *));      // Amp extra
71  void print_respawnable_list(job * list, void (*print)(job *));  // Amp extra
72
73  void terminal_signals(void (*func) (int));
74
75  void block_signal(int signal, int block);
76
77  // -----------------------------------------------------------------------
78  //      PUBLIC MACROS
79  // -----------------------------------------------------------------------
80
81  #define list_size(list)    list->pgid   // number of jobs in the list
82  #define empty_list(list)   !(list->pgid)  // returns 1 (true) if the list is empty
83
84  #define new_list(name)     new_job(0,name,FOREGROUND)  // name must be const char *
85
86  #define get_iterator(list)   list->next // return pointer to first job
87  #define has_next(iterator)   iterator
88  #define next(iterator)       ({job_iterator old = iterator; iterator = iterator->next;
    old;})
89
90  #define print_job_list(list)   print_list(list, print_item)
91  #define print_bg_job_list(list)   print_background_list(list, print_item)
92  #define print_stp_job_list(list)   print_stopped_list(list, print_item)
93  #define print_rsp_job_list(list)   print_respawnable_list(list, print_item)
94
95  #define restore_terminal_signals()   terminal_signals(SIG_DFL)
96  #define ignore_terminal_signals() terminal_signals(SIG_IGN)
97
98  #define set_terminal(pid)        tcsetpgrp (STDIN_FILENO,pid)
99  #define new_process_group(pid)   setpgid (pid, pid)
100
101 #define block_SIGCHLD()      block_signal(SIGCHLD, 1)
102 #define unblock_SIGCHLD()    block_signal(SIGCHLD, 0)
103
104 // macro for debugging----------------------------------------------------
105 // to debug integer i, use:    debug(i,%d);
106 // it will print out:  current line number, function name and file name, and also variable
    name, value and type
107 #define debug(x,fmt) fprintf(stderr,"\"%s\":%u:%s(): --> %s= " #fmt " (%s)\n", __FILE__,
    __LINE__, __FUNCTION__, #x, x, #fmt)
108
109 // -----------------------------------------------------------------------
110 #endif
111
112
```