**Check the following first:**

1. Assignment documentation (Assignment2.pdf) is provided
2. The project declaration is signed and included
3. The source code for at least one part is included together with its Makefile.

If any one of the above is missing, stop marking and award 0 mark.

If the self diagnosis and evaluation is not completed, the mark awarded cannot exceed 50.

|  |  |  |
| --- | --- | --- |
| **Item** | **Presentation and Documentation (10% of total)**: | **Feedback** |
|  | * The completed Project Declaration Form * List of files * Project title and description of the project * Filenames used in the submission. * Overall quality of the presentation.   Excellent: >8, good: 7-8. average: 4-6 |  |
| **Item** | **Part I - Basic functionalities (30%)**  Check whether each of 10 features listed below works or not. Allocate 3 marks if it works correctly with supporting evidence. Otherwise, award between 0 to 2 marks based on the strength of the test evidence presented and actual tests conducted. | **Feedback** |
| **1** | **Compilation and build:**  The source code can be built with the Makefile. |  |
| **2** | **Simple commands:**  Check whether simple commands work (each command line contains only one external command)  Eg,   * % ls * % ps * % ./show a b c * whether you can type and run simple command lines repeatedly   Note the command "show" is available from the Assignments page (tests.zip file) but you may need to recompile it. |  |
| **3** | **Shell Built-in:**  Check shell built-in commands "prompt", “pwd”, "cd" and "exit".  Eg:  % prompt myshell  % cd /tmp  % pwd  % cd (check whether it goes to the home directory)  % pwd  % cd ..  % pwd  % exit |  |
| **4** | **Tokenisation**  Check whether the shell can separate tokens in the command line.  Eg:  % ls -l -a (different number of spaces separating tokens)  % ls -l “aa bb cc” ‘cbd’ (quoted tokens containing spaces)  % show “-l” aa\&bb c\\d (tokens containing special characters) |  |
| **5** | **Long command:**  Check whether the shell can take a command with many command-line arguments  Eg:  % ls –l -t a b c  % show a bb ccc dddd 1 22 333 4444 555555  % show a b c d e f g h I j k l m n o p q r s t u v w x y z 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 20 abc123xyz |  |
| **6** | **Wildcards:**  Check whether the wildcards such as \* and ? can be expanded to matching file names. |  |
| **7** | **Handling Ctrl-C, Ctrl-\ and Ctrl-Z:**  Check whether the shell ignores SIGINT, SIGQUIT and SIGTSTP generated by typing Ctrl-C, Ctrl-\ and Ctrl-Z from shell prompt. |  |
| **8** | **Claim of zombies:**  You should check whether the shell has claimed zombie processes. One way to check this is: while running the shell in one terminal, open up another terminal. In the shell prompt, type the command "sleep 1 & sleep 2 & sleep 3 … ". In the other terminal, use the command "ps –elfH | grep your-userid" to check whether there are zombies (defunct children of the shell).  Use "whoami" to find your username and then use 'id your username' to find out your UID. |  |
| **9** | **Robustness of the program:**  Has the program ever crashed? How often it crashes? |  |
| **10** | **Quality of the code & algorithms**  Structure and readability of the code, conciseness and efficiency of the implementation |  |
| **Item** | **Part 1 – Advanced functionalities (30%)**  Check whether each of 10 features listed below works or not. Allocate 3 marks if it works correctly with supporting evidence. Otherwise, award between 0 to 2 marks based on the strength of the evidence presented and actual tests. | **Feedback** |
| **1** | **Sequential execution ";":**  Check whether the shell can execute a sequence of commands one after the other with “;” |  |
| **2** | **Concurrent execution "&":**  Check whether the shell can execute several commands concurrently with “&” |  |
| **3** | **Standard input, output and error redirections "<file", “>file” and “2>file”:**  Check whether the shell redirect the standard input, standard output and standard error to files |  |
| **4** | **Simple shell pipeline:**  Check whether the shell can send the output of the first command to the second command as the latter’s input using “|” |  |
| **5** | **Long shell pipeline:**  Check whether the shell can handle jobs with multiple pipelines (eg, 2 pipes, 3 pipes and 4 pipes in a single job) |  |
| **6** | **Shell history with shell built-in “history” and “!x”**  Check   * whether the shell can remember the history and the user can display the history with the shell built-in “history” and * whether the shell can repeat one of the previous command using “!xyz” where xyz represents the prefix of one of the previous command lines, such as “!ls” |  |
| **7** | **Shell history using arrow keys**  Check whether the user can display the previous command with the up arrow key and repeat the command by typing ENTER key. |  |
| **8** | **Complex command lines with multiple commands connected by “;” and “&”:**  Check whether the shell can handle complex command lines containing many commands connected by “;” and “&”. With No redirections but no pipe lines. |  |
| **9** | **Complex command lines with multiple jobs, some with redirections and some with pipelines. The command line may also contain wildcards.**  Check whether the shell can handle command lines that contain many jobs. Some of these jobs have redirections and some of these jobs have pipe lines. |  |
| **10** | **Handling of slow system calls:**  In the shell command loop where it reads a command line, a standard function such as fgets will be used. Ultimately these standard input functions would have to be implemented using the read system call. The blocking read can return prematurely when a signal is received (rather than when it receives an input). If such kind of event is not handled, the shell may behave erratically, such as suddenly displaying a shell prompt. It is a bit hard to check this feature, but I suggest that you can test the feature by sending a signal such as SIGCHLD to the shell to see whether it would change its behaviour. |  |
| **Item** | **Part 2 - Protocol specification (10%)**   * The specification must be complete with all necessary details, unambiguous (cannot be interpreted in any other way) and independent of implementation platforms and programming languages used (not limiting the implementation to any os platform and programming language). * The protocol must specify all three of the following: the format of the messages exchanged between the client and the server; the content of the messages exchanged between the client and the server; and the sequence of the messages exchanged between the client and the server. * The protocol must be able to do the required job logically.   Excellent: >8, good: 7-8. average: 4-6 | **Feedback** |
| **Item** | **Part 2 - Implementation of the client and the server (20%)**   * The client and server can be built using the Makefile * Authentication, at least with a test account “test/test” * The user can use the remote host via the client which is connected to the shell spawn by the server on the remote host, as if the user is interacting with the shell running locally. * the server can serve multiple clients concurrently   Excellent: >16, good: 14-16. average: 8-12 | **Feedback** |
|  | **Total: 100** |  |