Advanced Unix Programming
Lab 4

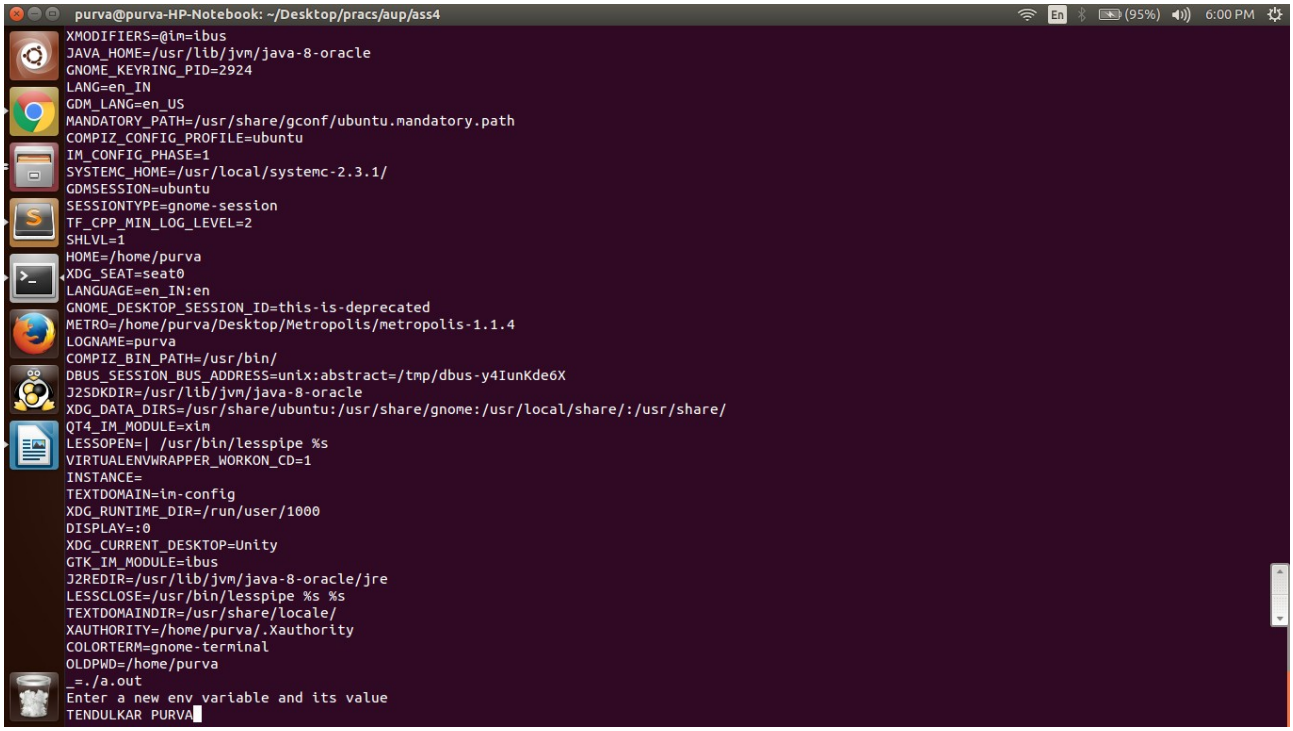
Purva Tendulkar : 111403049
Shruti Dogra : 111403075

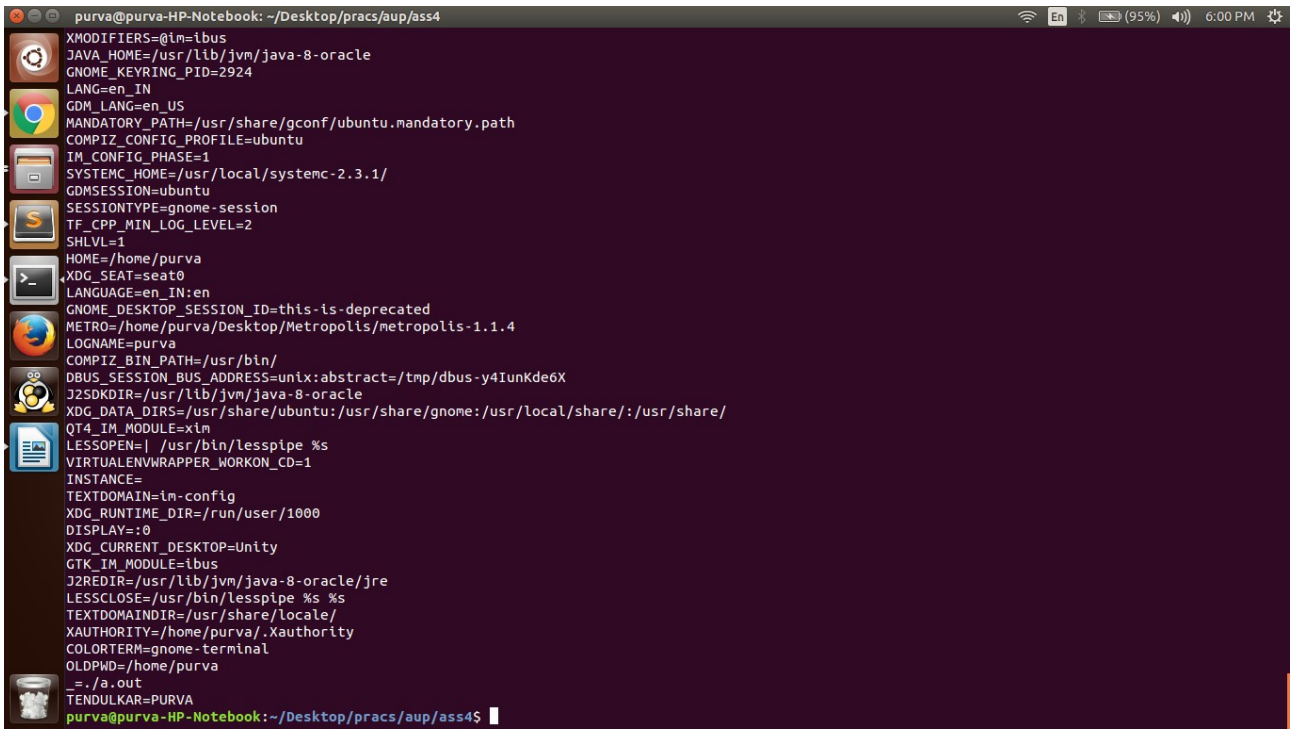
Q1. Print all existing environment variables with their values. Later input a new variable and its value and add to the environment list. Once again print the list.

Code :

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 20

extern char **environ;

int main() {
   char name[MAX], val[MAX], **env;

   for (env = environ; *env; ++env)
      printf("%s\n", *env);

   printf("Enter a new env variable and its value\n");
   scanf("%s%s", name, val);
   setenv(name, val, 1);

   for (env = environ; *env; ++env)
      printf("%s\n", *env);
   return 0;
}
```

Input and Output Screenshots :





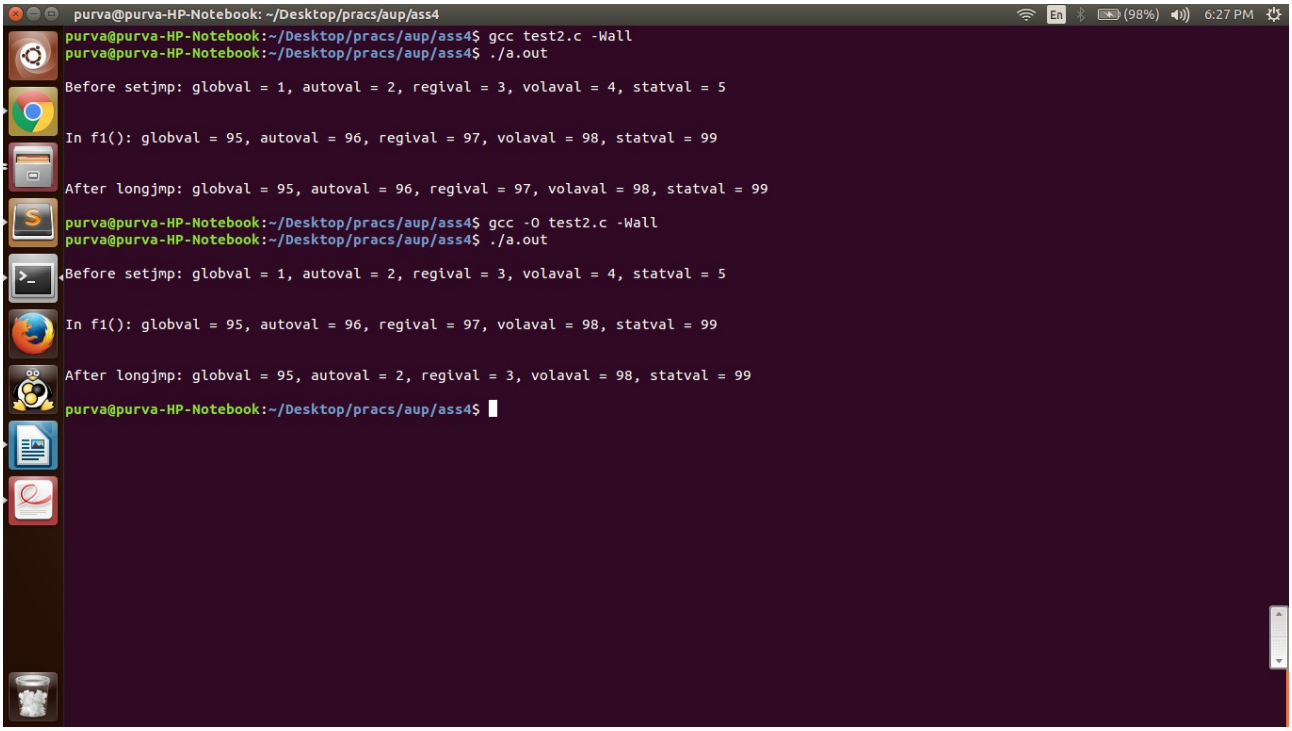
Explanation :
The new environment variable here is “TENDULKAR” and the corresponding value is “PURVA”.

Q2. With appropriate comments write a program using setjmp and longjmp to verify the status of different types of variables after invoking longjmp.
Code :

```
#include <setjmp.h>
#include <stdio.h>
#include <stdlib.h>

static int globval; //Global Variable
jmp_buf jmpbuffer;

void f2() {
	longjmp(jmpbuffer, 1);

	/* While invoking this longjmp, the variables that are stored in memory will have values
	as of the time of longjmp, while variables
	stored in registers are restored to their
	values when setjmp was called.
	Therefore values of autoval and regival get changed to 2 and 3 respectively whereas
	the rest remain the same */
}

void f1(int i, int j, int k, int l) {
	printf("\nIn f1(): ");
	printf("globval = %d, autoval = %d, regival = %d, volaval = %d, statval = %d\n\n", globval,
		i, j, k, l);

	/* In the above printf, the values of variables will be the same as passed to the function
	f1 as longjmp hasnt been invoked yet */
```

```
        f2();
}

int main() {
        int autoval;                    //Automatic Variable
        register int regival;           //Register Variable
        int volatile volaval;           //Volatile Variable
        static int statval;             //Static Variable
        globval = 1; autoval = 2; regival = 3; volaval = 4; statval = 5;

        /* The value of all types of variables are set before setjmp. When compiled using
        optimization (-O), AUTOMATIC and REGISTER variables get stored in the register. Rest
        get stored in memory. When compiled without optimization, all go to memory. */

        printf("\nBefore setjmp: ");
        printf("globval = %d, autoval = %d, regival = %d, volaval = %d, statval = %d\n\n",
                globval, autoval, regival, volaval, statval);

        if (setjmp(jmpbuffer) != 0) {
                printf("\nAfter longjmp: ");
                printf("globval = %d, autoval = %d, regival = %d, volaval = %d, statval = %d\n\n",
                globval, autoval, regival, volaval, statval);
                exit(0);
        }

        globval = 95, autoval = 96, regival = 97, volaval = 98, statval = 99;
        f1(autoval, regival, volaval, statval);
        exit(0);
}
```

Input and Output Screenshots :



Q3. Measures the performance of the getpid() and the fork functions using gettimeofday to measure the the execution time. Measure the performance ten times for each of the two system calls in the program itself and provide the timing results and compute an average for each system call.

Code :

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <time.h>
#include <unistd.h>
#define N 10

double get_time_difference(struct timeval start, struct timeval end) {
	return (end.tv_sec - start.tv_sec) + (end.tv_usec - start.tv_usec) / 1000000.0;
}

double time_getpid() {
	printf("getpid() function calls...\n");
	struct timeval getpid_start;
	struct timeval getpid_end;
	double sum_getpid_times_elapsed = 0.0;
	pid_t pid;
	int i;

	for(i = 0; i < N; i++){
		gettimeofday(&getpid_start, NULL);
		pid = getpid();
		gettimeofday(&getpid_end, NULL);
		sum_getpid_times_elapsed += get_time_difference(getpid_start, getpid_end);
		printf("%d - %lf\n", i, get_time_difference(getpid_start, getpid_end));
	}
	return sum_getpid_times_elapsed / 10;
}

double time_fork() {
	printf("fork() function calls...\n");
	struct timeval fork_start;
	struct timeval fork_end;
	double sum_fork_times_elapsed = 0.0;
	int i;

	for (i = 0; i < N; i++) {
		gettimeofday(&fork_start, NULL);
		int p = fork();
		gettimeofday(&fork_end, NULL);
		if (p) {
			sum_fork_times_elapsed += get_time_difference(fork_start, fork_end);
			printf("%d - %lf\n", i, get_time_difference(fork_start, fork_end));
		}
		else
			exit(0);
```

```
        }
        return sum_fork_times_elapsed / 10;
}


int main(){
        printf("-------------------------------------\n");
        printf("AVERAGE TIME FOR getpid() - %lf\n", time_getpid());
        printf("-------------------------------------\n");
        printf("AVERAGE TIME FOR fork() - %lf\n", time_fork());
        printf("-------------------------------------\n");
        return 0;
}
```

Input and Output Screenshots :