



# OS PROJECT – TASK MANAGER

Made by K213309, K213206, K214833

## Table of Contents

Introduction.....	2
Background .....	2
Platform and Languages .....	6
Methodology .....	7
Results.....	13
Problems & Challenges.....	19
Conclusion & Breakdown.....	19

## Introduction

Our motive was to recreate a “Process Monitor System” similar to the Windows “Task Manager” or Ubuntu “System Monitor”. This would include killing a process, setting the priority and affinity of a process, and monitoring CPU and MEM usage percentages.

Team Members:

1. Mohammad Yehya Hayati, K213309
2. Taha Ahmed, K214833
3. Sufyan Abdul Rasheed, K213206

## Background

The main research was done on how to implement different Operating System techniques to create a working Task Manager. To do this, we created 5 different header files so that we can accommodate our needs.

The 1<sup>st</sup> header file, MyMacros.h, consists of the inclusion of external header files and macro definitions.

```
1  #pragma once
2
3  #include <gtk/gtk.h>
4  #include <pthread.h>
5  #include <semaphore.h>
6  #include <sys/resource.h>
7  #include <sched.h>
8  #include <wait.h>
9  #define HARDWARE_CONCURRENCY (4)
10 #define GraphPoints 180
```

The 2<sup>nd</sup> header file, MyData.h, consists of the user defined structures and global variables.

```
1  #pragma once
2
3  #include "MyMacros.h"
4
5  typedef char BYTE;
6
7  typedef struct
8  {
9      GtkWidget *builder;
10     GtkWidget *window, *Fixed, *draw1, *draw2,
11     GtkWidget *PriorityPicker, *PriorityEntry,
12     GtkWidget *core[4];
13     GtkWidget *tv1;
14     GtkTreeSelection *tv1Selection;
15     GtkListStore *ProcessorData, *PriorityStore;
16     GtkTreeViewColumn *PID, *Name, *CPU, *MEM;
17     GtkCellRenderer *PID_, *Name_, *CPU_, *MEM_;
18     GtkTreeIter SelectedRow;
19     GtkTreeModel *model;
20
21     BYTE Affinity;
22     int PriorityLevel;
23     int TimerOn;
24     int cpuUtil, cpu[GraphPoints];
25     int memUtil[5], mem[5][GraphPoints];
26     double RGBGraph[5][3];
27 } TMDData;
28 TMDData TMD;
29
30 typedef struct
31 {
32     int *argc;
33     char ***argv;
34 } ARG;
35
36 sem_t ProcessMutex;
37 sem_t CPUMutex;
38 sem_t MEMMutex;
```

The 3<sup>rd</sup> header file, MyHashTable.h, consists of a modified hash table that we use to save process information and use with the GUI.

```
1  #include "MyData.h"
2  #define TableSize 300
3
4  typedef struct Node Node;
5  struct Node
6  {
13 void NodeConstructor(Node* x, char val[4][1024])
14 {
22 typedef struct HashTable
23 {
26 void HashTableConstructor(HashTable* a)
27 {
31 void HashTableDestructor(HashTable *a)
32 {
49 int Hash(char *value)
50 {
57 int Search(HashTable* a, char val[4][1024])
58 {
76 void Insert(HashTable* a, char val[4][1024])
77 {
94 void Delete(HashTable* a)
95 {
```

The 4<sup>th</sup> header file, MySignals.h, consists of all the signals that are used by GUI aspects like buttons and lists.

```
1  #pragma once
2
3  #include "MyData.h"
4
5  gboolean on_draw1_draw (GtkWidget *widget, cairo_t *cr, gpointer data)
6  {
45 gboolean on_draw2_draw (GtkWidget *widget, cairo_t *cr, gpointer data)
46 {
88 void on_tv1Selection_changed(GtkWidget *c)
89 {
99 void on_KillButton_clicked(GtkButton *b)
100 {
106 void on_PriorityButton_clicked(GtkButton *b)
107 {
113 void on_PriorityPicker_changed(GtkComboBox *c, GtkEntry *e)
114 {
117 void on_AffinityButton_clicked(GtkButton *b)
118 {
127 void on_cpu_toggled(GtkCheckButton *b)
128 {
```

The 5<sup>th</sup> header file, MyThreadFunctions.h, consists of the main driver functions which are also multithreaded to ensure optimal efficiency.

```
1  #pragma once
2
3  #include "MySignals.h"
4  #include "MyHashTable.h"
5
6  void Exit_Program()
7  {
11 void *ProcessHandler(void *args)
12 {
46 void *CpuHandler(void *args)
47 {
78 void *MemHandler(void *args)
79 {
115 void *Gui(void *args)
116 {
187 int TaskManagerStart(ARG *x)
188 {
```

We will further explain the code in the Methodology section.

## Platform and Languages

If it wasn't already obvious, our coding language is C. However, for the implementation of GUI, we used a C library called GTK+. In order to save time and effort, we used an IDE for creating a GUI called Glade.

Glade creates a .glade file which in essence is an .XML file which lets you define and store data in a shareable manner. Since GTK+ is compatible with Glade, it introduces some functions to import from a .glade file, which you can use to implement a GUI.

## Methodology

(For further inspection, see attached code)

First and foremost, we start from our C file which calls a function from MyThreadFunctions.h, TaskManagerStart ().

```
187 int TaskManagerStart(ARG *x)
188 {
189     sem_init(&ProcessMutex, 0, 0);
190     sem_init(&CPUMutex, 0, 0);
191     sem_init(&MEMMutex, 0, 0);
192     TMD.PriorityLevel = 0;
193     TMD.TimerOn = 1;
194     for(int i = 0 ; i < GraphPoints ; i++) TMD.cpu[i] = 0;
195     for(int i = 0 ; i < 5 ; i++) for(int j = 0 ; j < GraphPoints ; j++) TMD.mem[i][j] = 0;
196     double temp[5][3] = {{0.0,0.7,0.7},{1,0,0},{0,1,0},{1,1,0},{1,0.5,0}};
197     for(int i = 0 ; i < 5 ; i++) for(int j = 0 ; j < 3 ; j++) TMD.RGBGraph[i][j] = temp[i][j];
198     pthread_t GuiThread;
199     pthread_create(&GuiThread, NULL, Gui, (void*)x);
200     pthread_join(GuiThread, NULL);
201     return 0;
202 }
```

All this function does is set standard data members (non-GTK+) and starts a new thread called GuiThread which execute the GUI function, which will perform the main work.

Since the code for the GUI function is too large to insert, we will leave it to you to explore the attached code. The GUI function sets all GTK+ data members from the .XML file. Once they are set, it creates 3 threads each of which perform different work. The three threads execute the function CpuHandler, MemHandler, and ProcessHandler.



The CpuHandler function implements the code to calculate the average cpu usage percentage every 1 second. This is done by reading a system file called /proc/stat. Once the data is read, some string and array manipulation is performed and a signal is sent to a GTK+ object called draw, which does exactly what it sounds like; draw.

```
46 void *CpuHandler(void *args)
47 {
48     sem_wait(&CPUMutex);
49
50     while(TMD.TimerOn)
51     {
52         static long time1 = 0, time2 = 0;
53         static int flag = 0;
54         char line[128], dummy[32];
55
56         FILE *p1 = fopen("/proc/stat", "r");
57         fgets(line, 128, p1);
58         sscanf(line, "%s %ld", dummy, &time2);
59         fclose(p1);
60
61         if (!flag)
62         {
63             flag = 1;
64             time1 = time2;
65             sleep(1);
66             continue;
67         }
68
69         TMD.cpuUtil = time2 - time1;
70         for (int i = 0; i < GraphPoints-1; i++) TMD.cpu[i] = TMD.cpu[i+1];
71         TMD.cpu[GraphPoints-1] = TMD.cpuUtil/(HARDWARE_CONCURRENCY);
72         time1 = time2;
73         gtk_widget_queue_draw(TMD.draw1);
74         sleep(1);
75     }
76     pthread_exit(0);
77 }
```

The results of the draw section will be shown the results section.

The MemHandler function performs fairly similarly to the CpuHandler function. However, instead of reading from /proc/stat, it reads from /proc/meminfo. Also we read 4 different values which we also draw on a grid.

```
78 void *MemHandler(void *args)
79 {
80     sem_wait(&HEMMutex);
81
82     while(TMD.TimerOn)
83     {
84         static long time1[5] = {0}, time2[5] = {0};
85         static int flag = 0;
86         char line[128], dummy[32];
87
88         FILE *p1 = fopen("/proc/meminfo", "r");
89         for(int i = 0 ; i < 5 ; i++)
90         {
91             fgets(line, 128, p1);
92             sscanf(line, "%[^0-9] %ld", dummy, &time2[i]);
93         }
94         fclose(p1);
95
96         if (!flag)
97         {
98             flag = 1;
99             for(int i = 0 ; i < 5 ; i++) time1[i] = time2[i];
100             sleep(1);
101             continue;
102         }
103         for(int i = 1 ; i < 5 ; i++)
104         {
105             TMD.memUtil[i] = time2[i] * 100 / time1[0];
106             for (int j = 0; j < GraphPoints-1 ; j++) TMD.mem[i][j] = TMD.mem[i][j+1];
107             TMD.mem[i][GraphPoints-1] = TMD.memUtil[i];
108             time1[i] = time2[i];
109         }
110         gtk_widget_queue_draw(TMD.draw2);
111         sleep(1);
112     }
113     pthread_exit(0);
114 }
```

The ProcessHandler function is bit different from the prior functions. It calculates and stores the PID, CPU%, MEM%, and Name of each running process in a hash table. It then puts this data into a GtkListStore which is a built-in linked list for the GTK+ library. This is done by using a system call called `execlp()`. We run a bash script which logs all the processes in a txt file, and then we read the txt file and manipulate the data to that we can store the data. After this, the data is ready for display, and is updated every second.

```

11 void *ProcessHandler(void *args)
12 {
13     sem_wait(&ProcessMutex);
14
15     char *vals[] = {NULL, "--sort=pid", "--sort=%cpu", "--sort=%mem", "--sort=command"};
16     char line[1024] = {0}, Info[4][1024] = {0};
17     gtk_list_store_clear(TMD.ProcessorData);
18     HashTable HT;
19     HashTableConstructor(&HT);
20     while(TMD.TimerOn)
21     {
22         if(!fork()) execlp("/bin/sh", "/bin/sh", "ProcessInfo.sh", vals[2], NULL);
23         else wait(NULL);
24         FILE *fptr = fopen("Processes.txt", "r");
25         int x = 0;
26         GtkTreeIter iter;
27         while(TMD.TimerOn)
28         {
29             if (!fgets(line, 1024, fptr) || !TMD.TimerOn)
30             {
31                 fclose(fptr);
32                 break;
33             }
34             if(!x++) continue;
35             int check = sscanf(line, "%s %s %s %s\n", Info[0], Info[1], Info[2], Info[3]);
36             if(TMD.TimerOn && Search(&HT, Info) == -1) Insert(&HT, Info);
37         }
38         if(TMD.TimerOn)
39         {
40             sleep(1);
41             Delete(&HT);
42         }
43     }
44     HashTableDestructor(&HT);
45 }

```

However, if we just let the ProcessHandler function run like that, the list would just keep on getting appended over and over again. Therefore, we need to manipulate the data in the hash table which would in turn reflect over to the GtkListStore. This done by using two hash table functions called Search () and Insert ().

```
57 int Search(HashTable* a, char val[4][1024])
58 {
59     int h = Hash(val[3]);
60     Node* temp = a->Table[h];
61     while(temp != NULL)
62     {
63         if(!strcmp(temp->name, val[3]))
64         {
65             if(gtk_list_store_iter_is_valid(GTK_LIST_STORE(TMD.ProcessorData), &temp->iter))
66             {
67                 gtk_list_store_set(GTK_LIST_STORE(TMD.ProcessorData), &temp->iter, 0, val[0], 1, val[1], 2, val[2], 3, val[3], -1);
68                 temp->flag = 1;
69                 return h;
70             }
71             temp = temp->next;
72         }
73     }
74     return -1;
75 }
76 void Insert(HashTable* a, char val[4][1024])
77 {
78     int h = Hash(val[3]);
79     Node* newnode = (Node*)malloc(sizeof(Node));
80     if(TMD.TimerOn)
81     {
82         NodeConstructor(newnode, val);
83         if (a->Table[h] == NULL) a->Table[h] = newnode;
84         else
85         {
86             Node* temp = a->Table[h];
87             while (temp->next != NULL) temp = temp->next;
88             temp->next = newnode;
89             newnode->pre = temp;
90         }
91     }
92     else free(newnode);
93 }
```

There are also some GTK+ objects defined to implement some features like killing a process using a button, setting priority using a combo box, setting processor affinity by using check boxes. To implement this procedure, GTK+ uses its own built-in signal system, therefore some signals were defined.

When the kill button is pressed,

```
99 void on_KillButton_clicked(GtkButton *b)
100 {
101     if(!GTK_IS_TREE_MODEL(TMD.model))return;
102     gchar *value;
103     gtk_tree_model_get(TMD.model, &TMD.SelectedRow, 0, &value, -1);
104     if(!kill(atoi(value), SIGKILL)) gtk_list_store_remove(GTK_LIST_STORE(TMD.ProcessorData), &TMD.SelectedRow);
105 }
```

When the priority button is pressed and when the combo box is opened,

```
106 void on_PriorityButton_clicked(GtkButton *b)
107 {
108     if(!GTK_IS_TREE_MODEL(TMD.model))return;
109     gchar *value;
110     gtk_tree_model_get(TMD.model, &TMD.SelectedRow, 0, &value, -1);
111     setpriority(PRIO_PROCESS, atoi(value), TMD.PriorityLevel);
112 }
113 void on_PriorityPicker_changed(GtkComboBox *c, GtkEntry *e)
114 {
115     TMD.PriorityLevel = atoi(gtk_entry_get_text(e));
116 }
```

When the affinity button is pressed and when the check boxes are checked,

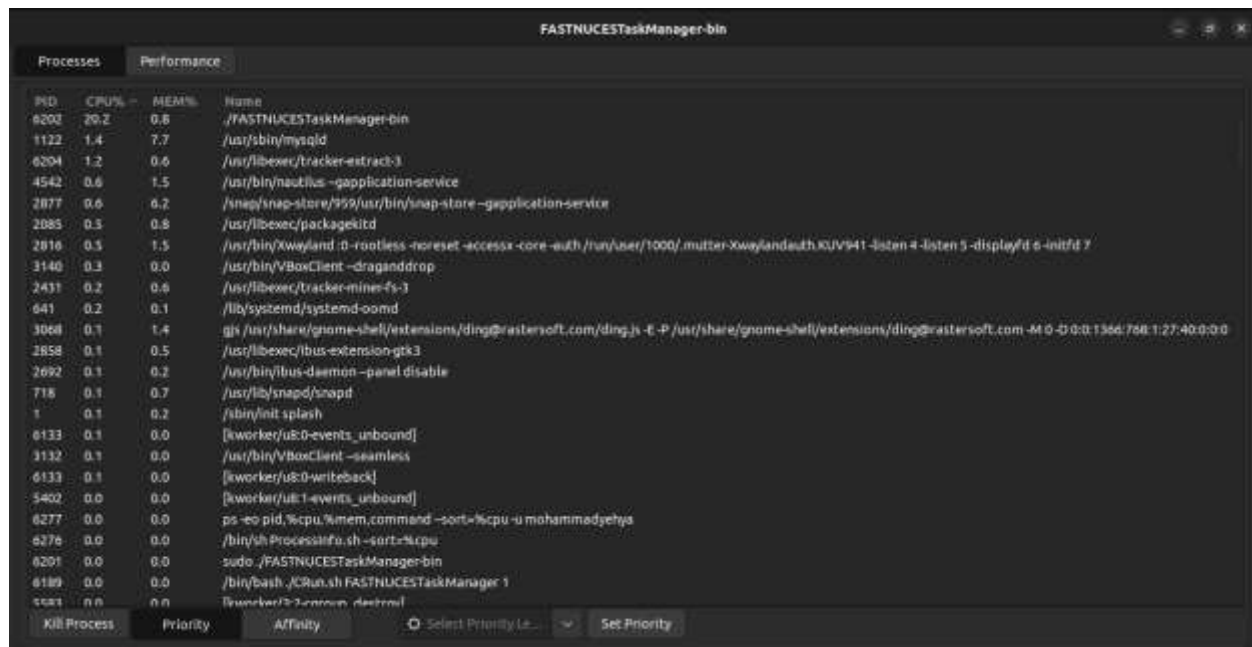
```
117 void on_AffinityButton_clicked(GtkButton *b)
118 {
119     if(!GTK_IS_TREE_MODEL(TMD.model))return;
120     gchar *value;
121     cpu_set_t mask;
122     CPU_ZERO(&mask);
123     for(int i = 0 ; i < 4 ; i++) if((TMD.Affinity & 1 << i) == 1 << i) CPU_SET(i, &mask);
124     gtk_tree_model_get(TMD.model, &TMD.SelectedRow, 0, &value, -1);
125     sched_setaffinity(atoi(value), sizeof(mask), &mask);
126 }
127 void on_cpu_toggled(GtkCheckButton *b)
128 {
129     int x = 0;
130     sscanf(gtk_widget_get_name((GtkWidget*)b), "cpu%d", &x);
131     TMD.Affinity ^= (1 << x);
132 }
```

(For further inspection, see attached code)

## Results

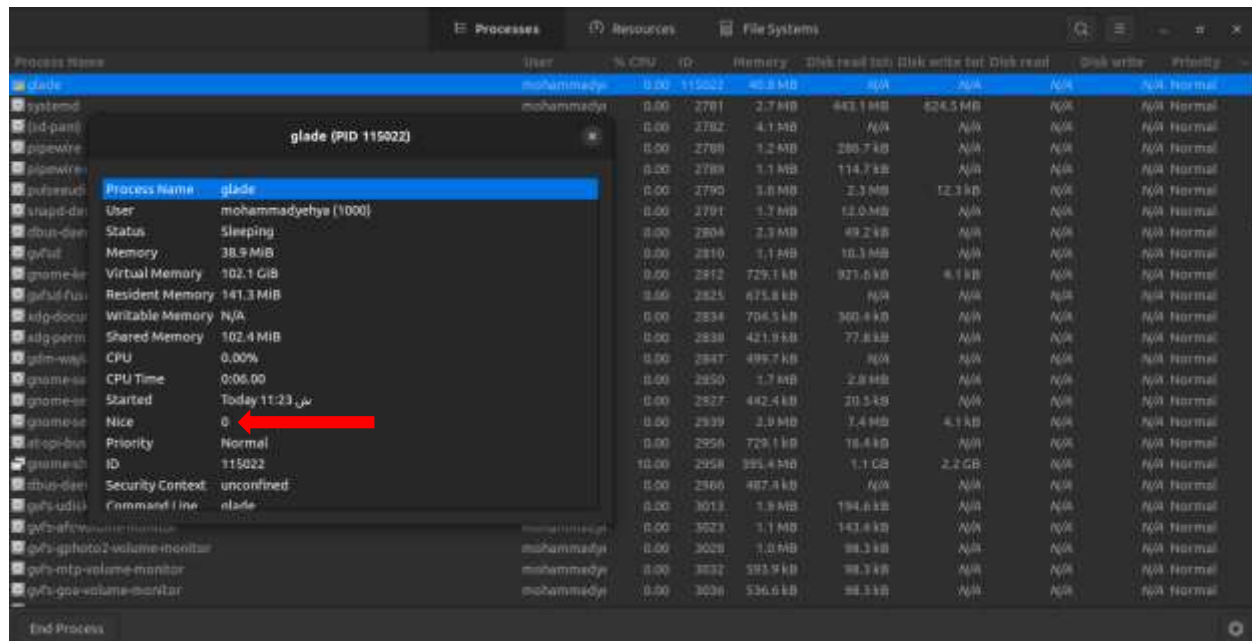
In the end, when we combine all these theories and logic and mix them up with some programming skill we are left with a fully functional Task Manager. Here are some of the results. To check whether our application is properly setting priorities and affinities, we will compare with Ubuntu's built-in System Monitor.

The first look at the project running.



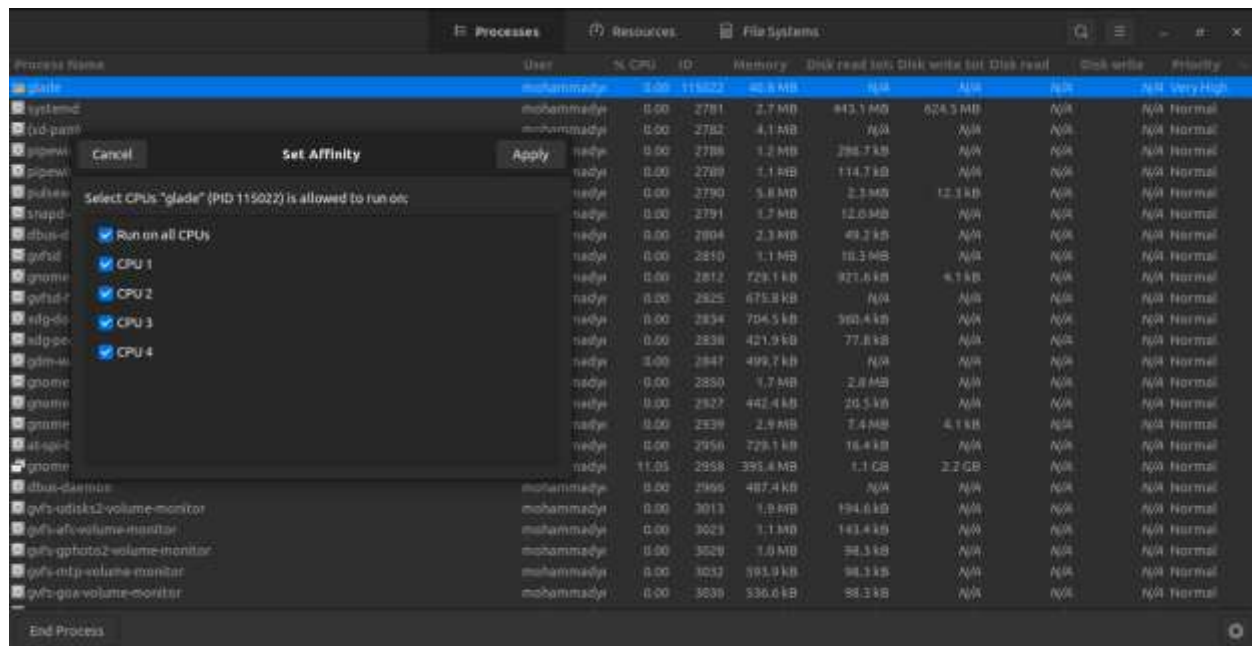
PID	CPU%	MEM%	Name
6202	20.2	0.8	/FASTNUCESTaskManager-bin
1122	1.4	7.7	/usr/sbin/mysqld
6204	1.2	0.6	/usr/libexec/tracker-extract-3
4542	0.6	1.5	/usr/bin/nautilus --application-service
2877	0.6	0.2	/snap/snap-store/959/usr/bin/snap-store --application-service
2085	0.3	0.8	/usr/libexec/packagekitd
2816	0.3	1.5	/usr/bin/Xwayland -D -rootless -noreset -accessx -core -auth /run/user/1000/.mutter-Xwaylandauth.KUV941 -listen 4 -listen 5 -displayfd 6 -initfd 7
3140	0.3	0.0	/usr/bin/VBoxClient --draganddrop
2431	0.2	0.6	/usr/libexec/tracker-mime-fs-3
641	0.2	0.1	/lib/systemd/systemd-oomd
3068	0.1	1.4	gjs /usr/share/gnome-shell/extensions/ding@rastersoft.com/ding.js -E -P /usr/share/gnome-shell/extensions/ding@rastersoft.com -M 0 -D 0:0:136d:758:1:27:40:0:0:0
2858	0.1	0.5	/usr/libexec/ibus-extension-gtk3
2692	0.1	0.2	/usr/bin/ibus-daemon --panel disable
718	0.1	0.7	/usr/lib/snapd/snapd
1	0.1	0.2	/sbin/init splash
6133	0.1	0.0	[kworker/u8:0-events_unbound]
3132	0.1	0.0	/usr/bin/VBoxClient --seamless
6133	0.1	0.0	[kworker/u8:0-writeback]
5402	0.0	0.0	[kworker/u8:1-events_unbound]
6277	0.0	0.0	ps -eo pid,%cpu,%mem,command --sort=%cpu -u mohammadyehya
6276	0.0	0.0	/bin/sh ProcessInfo.sh --sort=%cpu
6201	0.0	0.0	sudo ./FASTNUCESTaskManager-bin
6189	0.0	0.0	/bin/bash ./CRun.sh FASTNUCESTaskManager 1
6081	0.0	0.0	[kworker/3:2-mmio_interrupt]

Before setting the priority. (Checked via System Monitor)



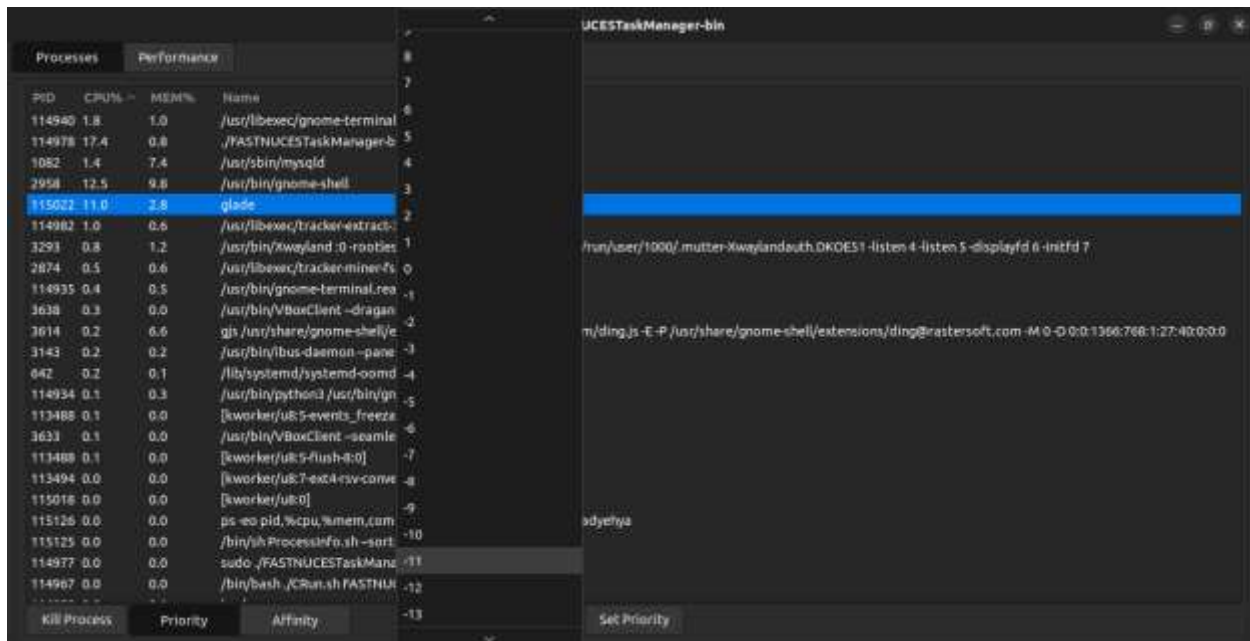
Process Name	User	% CPU	ID	Memory	Disk read tot	Disk write tot	Disk read	Disk write	Priority
glade	mohammadye	0.00	115022	40.8 MB	N/A	N/A	N/A	N/A	Normal
systemd	mohammadye	0.00	2781	2.7 MB	443.1 MB	624.5 MB	N/A	N/A	Normal
(xid-pant)	mohammadye	0.00	2782	4.1 MB	N/A	N/A	N/A	N/A	Normal
pipewire	mohammadye	0.00	2788	1.2 MB	280.7 kB	N/A	N/A	N/A	Normal
pipewire	mohammadye	0.00	2789	1.1 MB	114.7 kB	N/A	N/A	N/A	Normal
pulseaudio	mohammadye	0.00	2790	3.8 MB	2.3 MB	12.3 kB	N/A	N/A	Normal
strawd-da	mohammadye (1000)	0.00	2791	1.7 MB	12.0 MB	N/A	N/A	N/A	Normal
dbus-daem	mohammadye	0.00	2804	2.3 MB	49.2 kB	N/A	N/A	N/A	Normal
gvfs	mohammadye	0.00	2810	1.1 MB	10.3 MB	N/A	N/A	N/A	Normal
gnome-ke	mohammadye	0.00	2812	729.1 kB	921.6 kB	4.1 kB	N/A	N/A	Normal
gvfs-fu	mohammadye	0.00	2825	675.8 kB	N/A	N/A	N/A	N/A	Normal
kidg-docu	mohammadye	0.00	2834	704.5 kB	300.4 kB	N/A	N/A	N/A	Normal
kidg-perm	mohammadye	0.00	2838	421.9 kB	77.8 kB	N/A	N/A	N/A	Normal
gdm-wa	mohammadye	0.00	2847	499.7 kB	N/A	N/A	N/A	N/A	Normal
gnome-ss	mohammadye	0.00	2850	1.7 MB	2.8 MB	N/A	N/A	N/A	Normal
gnome-ss	mohammadye	0.00	2857	442.4 kB	20.5 kB	N/A	N/A	N/A	Normal
gnome-ss	mohammadye	0.00	2839	2.9 MB	7.4 MB	4.1 kB	N/A	N/A	Normal
at-spi-bus	mohammadye	0.00	2956	729.1 kB	18.4 kB	N/A	N/A	N/A	Normal
gnome-sh	mohammadye	10.00	2958	395.4 MB	1.1 GB	2.2 GB	N/A	N/A	Normal
dbus-daem	mohammadye	0.00	2966	487.4 kB	N/A	N/A	N/A	N/A	Normal
gvfs-udis	mohammadye	0.00	3013	1.8 MB	194.6 kB	N/A	N/A	N/A	Normal
gvfs-afn	mohammadye	0.00	3023	1.1 MB	143.4 kB	N/A	N/A	N/A	Normal
gvfs-gphoto2-volume-monitor	mohammadye	0.00	3028	1.0 MB	38.3 kB	N/A	N/A	N/A	Normal
gvfs-mtp-volume-monitor	mohammadye	0.00	3032	983.9 kB	98.3 kB	N/A	N/A	N/A	Normal
gvfs-goa-volume-monitor	mohammadye	0.00	3036	536.6 kB	98.3 kB	N/A	N/A	N/A	Normal

Before setting the affinity. (Checked via System Monitor)

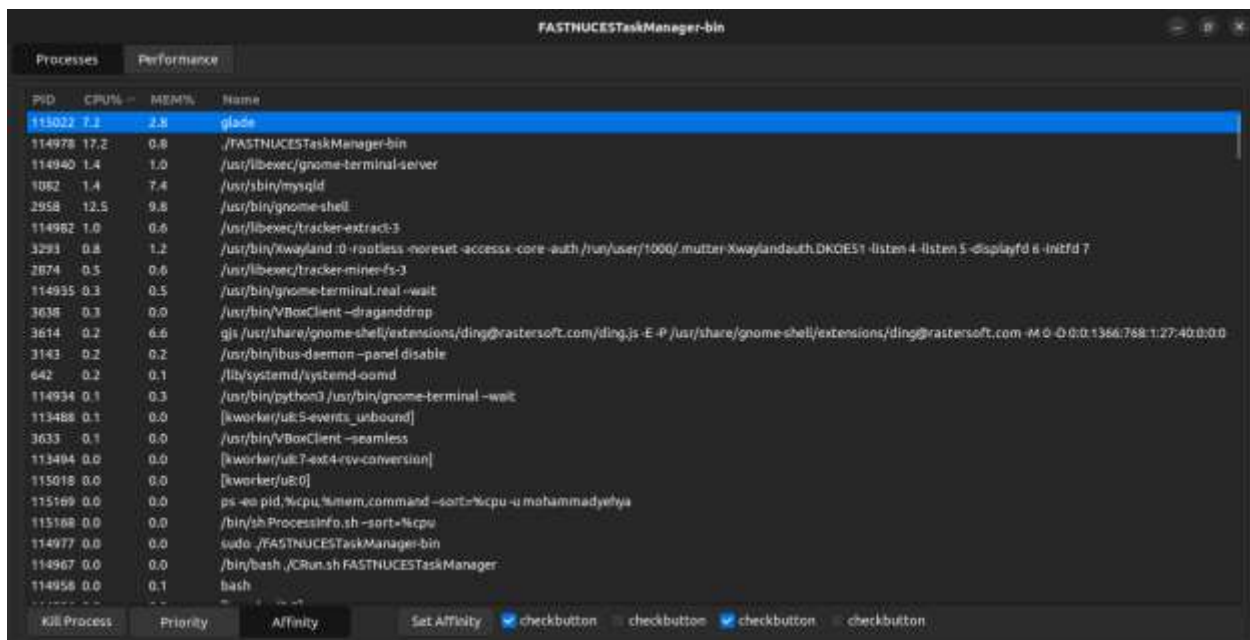


Process Name	User	% CPU	ID	Memory	Disk read tot	Disk write tot	Disk read	Disk write	Priority
glade	mohammadye	0.00	115022	40.8 MB	N/A	N/A	N/A	N/A	Very High
systemd	mohammadye	0.00	2781	2.7 MB	443.1 MB	624.5 MB	N/A	N/A	Normal
(xid-pant)	mohammadye	0.00	2782	4.1 MB	N/A	N/A	N/A	N/A	Normal
pipewire	mohammadye	0.00	2788	1.2 MB	280.7 kB	N/A	N/A	N/A	Normal
pipewire	mohammadye	0.00	2789	1.1 MB	114.7 kB	N/A	N/A	N/A	Normal
pulseaudio	mohammadye	0.00	2790	3.8 MB	2.3 MB	12.3 kB	N/A	N/A	Normal
strawd-da	mohammadye	0.00	2791	1.7 MB	12.0 MB	N/A	N/A	N/A	Normal
dbus-daem	mohammadye	0.00	2804	2.3 MB	49.2 kB	N/A	N/A	N/A	Normal
gvfs	mohammadye	0.00	2810	1.1 MB	10.3 MB	N/A	N/A	N/A	Normal
gnome-ke	mohammadye	0.00	2812	729.1 kB	921.6 kB	4.1 kB	N/A	N/A	Normal
gvfs-fu	mohammadye	0.00	2825	675.8 kB	N/A	N/A	N/A	N/A	Normal
kidg-docu	mohammadye	0.00	2834	704.5 kB	300.4 kB	N/A	N/A	N/A	Normal
kidg-perm	mohammadye	0.00	2838	421.9 kB	77.8 kB	N/A	N/A	N/A	Normal
gdm-wa	mohammadye	0.00	2847	499.7 kB	N/A	N/A	N/A	N/A	Normal
gnome-ss	mohammadye	0.00	2850	1.7 MB	2.8 MB	N/A	N/A	N/A	Normal
gnome-ss	mohammadye	0.00	2857	442.4 kB	20.5 kB	N/A	N/A	N/A	Normal
gnome-ss	mohammadye	0.00	2839	2.9 MB	7.4 MB	4.1 kB	N/A	N/A	Normal
at-spi-bus	mohammadye	0.00	2956	729.1 kB	18.4 kB	N/A	N/A	N/A	Normal
gnome-sh	mohammadye	11.00	2958	395.4 MB	1.1 GB	2.2 GB	N/A	N/A	Normal
dbus-daem	mohammadye	0.00	2966	487.4 kB	N/A	N/A	N/A	N/A	Normal
gvfs-udis	mohammadye	0.00	3013	1.8 MB	194.6 kB	N/A	N/A	N/A	Normal
gvfs-afn	mohammadye	0.00	3023	1.1 MB	143.4 kB	N/A	N/A	N/A	Normal
gvfs-gphoto2-volume-monitor	mohammadye	0.00	3028	1.0 MB	38.3 kB	N/A	N/A	N/A	Normal
gvfs-mtp-volume-monitor	mohammadye	0.00	3032	983.9 kB	98.3 kB	N/A	N/A	N/A	Normal
gvfs-goa-volume-monitor	mohammadye	0.00	3036	536.6 kB	98.3 kB	N/A	N/A	N/A	Normal

Setting the priority of a process.

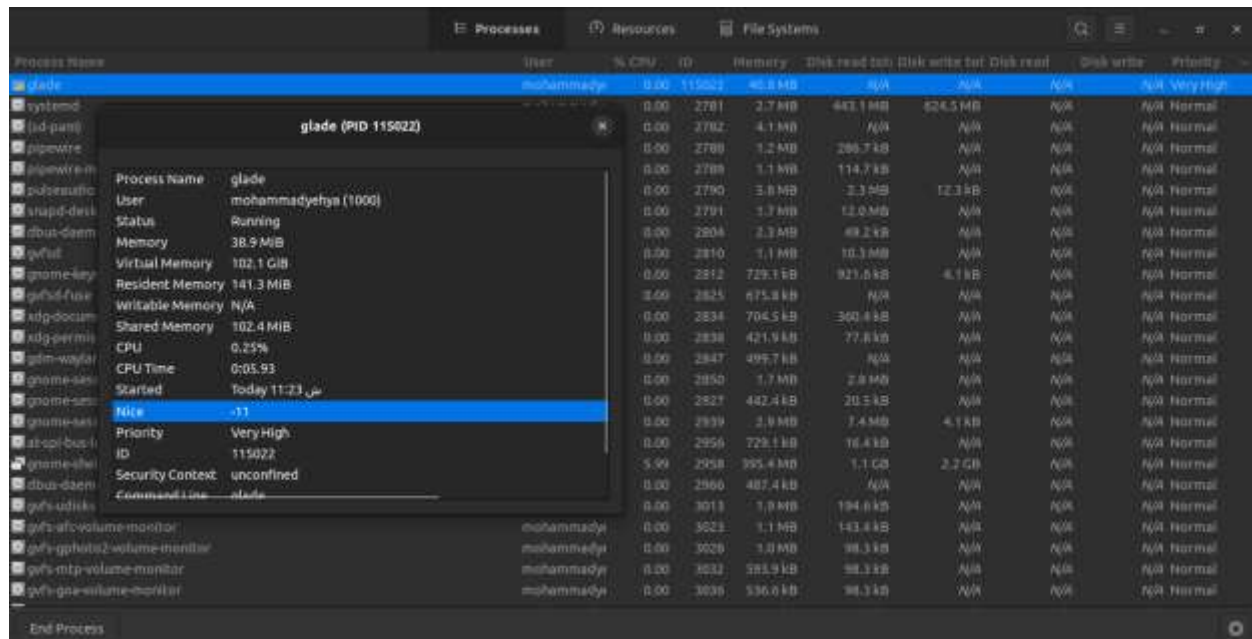


Setting the affinity of a process.





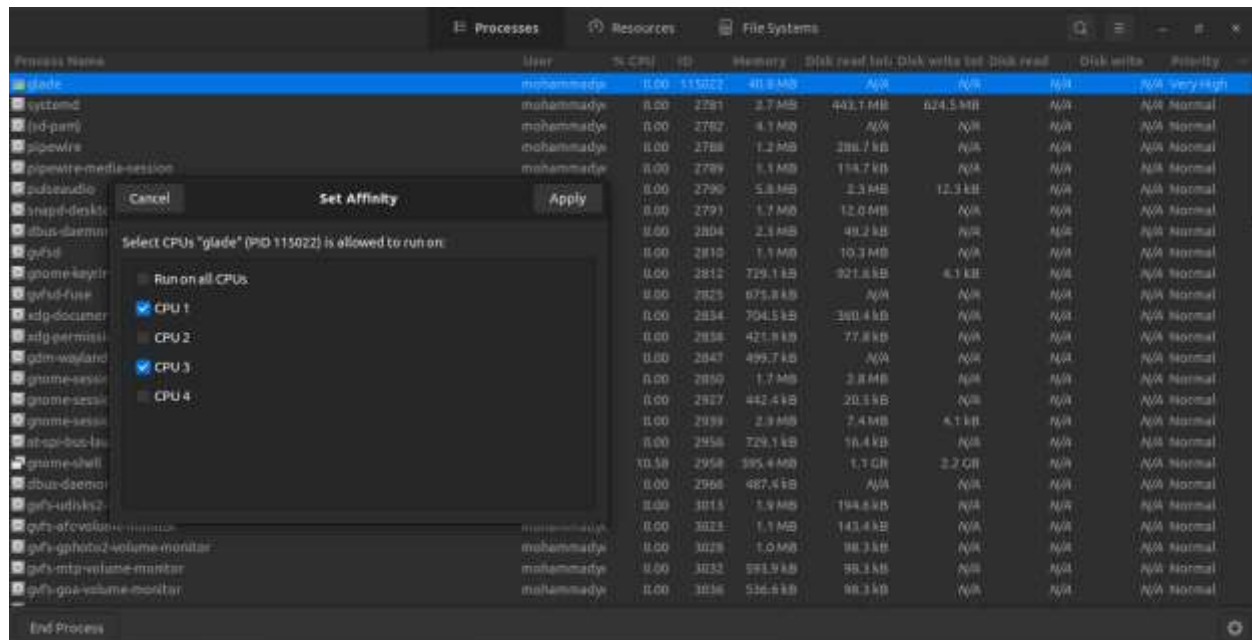
After setting the priority. (Checked via System Monitor)



The screenshot shows the System Monitor application with the 'Processes' tab selected. A window titled 'glade (PID 115022)' is open, displaying various system metrics for the 'glade' process. The process is running under the user 'mohammadyehya (1000)' with a priority of 'Very High'. The window also shows the process's memory usage, virtual memory, and resident memory.

Process Name	User	% CPU	ID	Memory	Disk read tot	Disk write tot	Disk read	Disk write	Priority
glade	mohammadyehya	0.00	115022	40.8 MB	N/A	N/A	N/A	N/A	Very High
systemd	mohammadyehya	0.00	2781	2.7 MB	443.1 MB	624.5 MB	N/A	N/A	Normal
(td-pam)	mohammadyehya	0.00	2782	4.1 MB	N/A	N/A	N/A	N/A	Normal
pipewire	mohammadyehya	0.00	2788	1.2 MB	286.7 kB	N/A	N/A	N/A	Normal
pipewire-media-session	mohammadyehya	0.00	2789	1.1 MB	114.7 kB	N/A	N/A	N/A	Normal
pulseaudio	mohammadyehya	0.00	2790	3.8 MB	2.3 MB	12.3 kB	N/A	N/A	Normal
snapsd-desktop	mohammadyehya	0.00	2791	1.7 MB	12.0 MB	N/A	N/A	N/A	Normal
dbus-daemon	mohammadyehya	0.00	2804	2.3 MB	49.2 kB	N/A	N/A	N/A	Normal
gvfsd	mohammadyehya	0.00	2810	1.1 MB	10.3 MB	N/A	N/A	N/A	Normal
gnome-keyring	mohammadyehya	0.00	2812	729.1 kB	921.6 kB	4.1 kB	N/A	N/A	Normal
gvfsd-fuse	mohammadyehya	3.60	2825	875.8 kB	N/A	N/A	N/A	N/A	Normal
xdg-documents	mohammadyehya	0.00	2834	704.5 kB	360.4 kB	N/A	N/A	N/A	Normal
xdg-permissions	mohammadyehya	0.00	2838	421.9 kB	77.8 kB	N/A	N/A	N/A	Normal
gdm-wayland	mohammadyehya	0.00	2847	499.7 kB	N/A	N/A	N/A	N/A	Normal
gnome-session	mohammadyehya	0.00	2850	1.7 MB	2.8 MB	N/A	N/A	N/A	Normal
gnome-session	mohammadyehya	0.00	2857	442.4 kB	20.3 kB	N/A	N/A	N/A	Normal
gnome-session	mohammadyehya	0.00	2939	2.9 MB	7.4 MB	4.1 kB	N/A	N/A	Normal
atspi-bus-daemon	mohammadyehya	0.00	2956	729.1 kB	16.4 kB	N/A	N/A	N/A	Normal
gnome-shell	mohammadyehya	10.38	2958	395.4 MB	1.1 GB	2.2 GB	N/A	N/A	Normal
dbus-daemon	mohammadyehya	0.00	2966	487.4 kB	N/A	N/A	N/A	N/A	Normal
gvfs-udisks2	mohammadyehya	0.00	3013	1.9 MB	194.8 kB	N/A	N/A	N/A	Normal
gvfs-goa-volume-monitor	mohammadyehya	0.00	3023	1.1 MB	143.4 kB	N/A	N/A	N/A	Normal
gvfs-gphoto2-volume-monitor	mohammadyehya	0.00	3028	1.0 MB	98.3 kB	N/A	N/A	N/A	Normal
gvfs-mtp-volume-monitor	mohammadyehya	0.00	3032	383.9 kB	98.3 kB	N/A	N/A	N/A	Normal
gvfs-goa-volume-monitor	mohammadyehya	0.00	3036	536.0 kB	98.3 kB	N/A	N/A	N/A	Normal

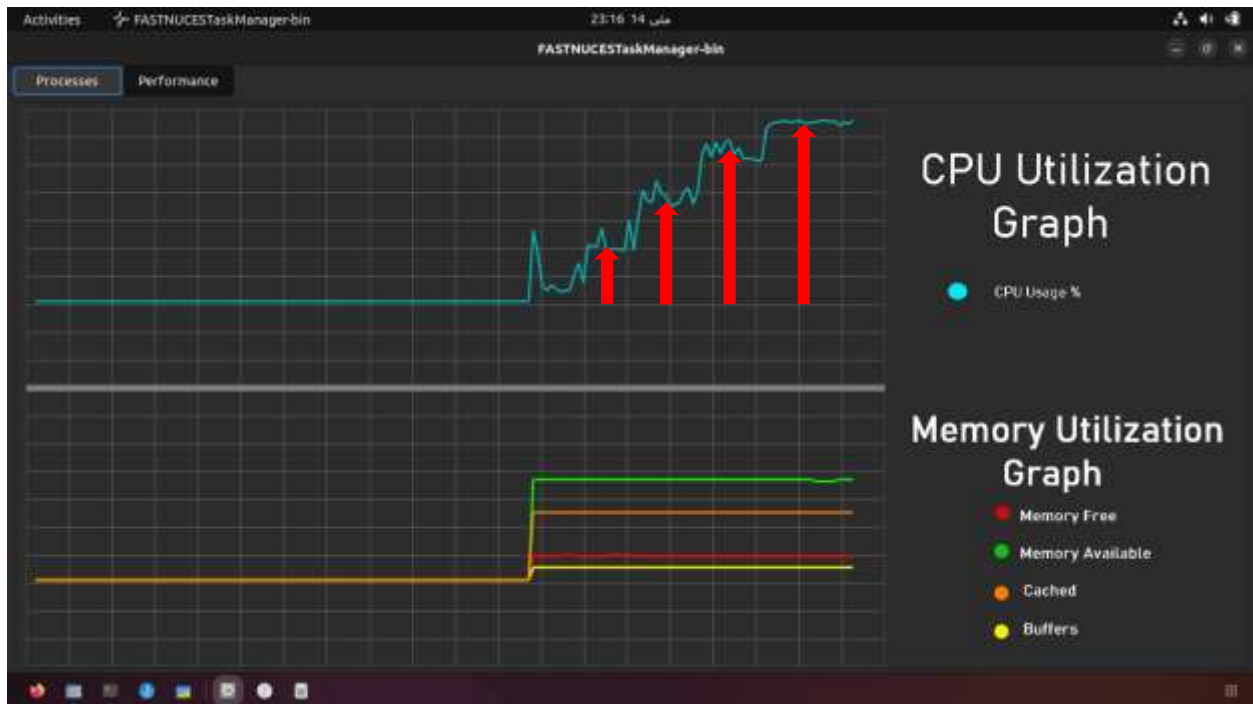
After setting the affinity. (Checked via System Monitor)



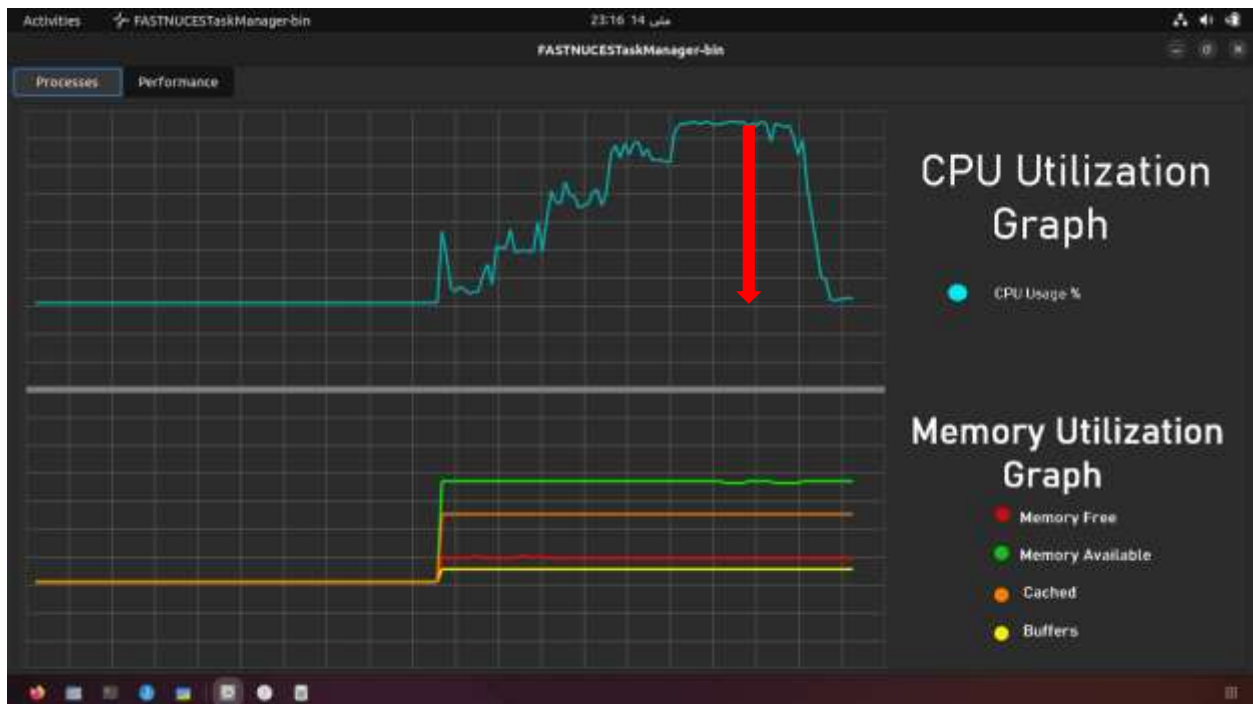
The screenshot shows the System Monitor application with the 'Processes' tab selected. A window titled 'Set Affinity' is open, allowing the user to select the CPUs that the 'glade' process (PID 115022) is allowed to run on. The window shows a list of CPUs (CPU 1, CPU 2, CPU 3, CPU 4) and a checkbox for 'Run on all CPUs'. The 'glade' process is currently running on CPU 1.

Process Name	User	% CPU	ID	Memory	Disk read tot	Disk write tot	Disk read	Disk write	Priority
glade	mohammadyehya	0.00	115022	40.8 MB	N/A	N/A	N/A	N/A	Very High
systemd	mohammadyehya	0.00	2781	2.7 MB	443.1 MB	624.5 MB	N/A	N/A	Normal
(td-pam)	mohammadyehya	0.00	2782	4.1 MB	N/A	N/A	N/A	N/A	Normal
pipewire	mohammadyehya	0.00	2788	1.2 MB	286.7 kB	N/A	N/A	N/A	Normal
pipewire-media-session	mohammadyehya	0.00	2789	1.1 MB	114.7 kB	N/A	N/A	N/A	Normal
pulseaudio	mohammadyehya	0.00	2790	3.8 MB	2.3 MB	12.3 kB	N/A	N/A	Normal
snapsd-desktop	mohammadyehya	0.00	2791	1.7 MB	12.0 MB	N/A	N/A	N/A	Normal
dbus-daemon	mohammadyehya	0.00	2804	2.3 MB	49.2 kB	N/A	N/A	N/A	Normal
gvfsd	mohammadyehya	0.00	2810	1.1 MB	10.3 MB	N/A	N/A	N/A	Normal
gnome-keyring	mohammadyehya	0.00	2812	729.1 kB	921.6 kB	4.1 kB	N/A	N/A	Normal
gvfsd-fuse	mohammadyehya	0.00	2825	875.8 kB	N/A	N/A	N/A	N/A	Normal
xdg-documents	mohammadyehya	0.00	2834	704.5 kB	360.4 kB	N/A	N/A	N/A	Normal
xdg-permissions	mohammadyehya	0.00	2838	421.9 kB	77.8 kB	N/A	N/A	N/A	Normal
gdm-wayland	mohammadyehya	0.00	2847	499.7 kB	N/A	N/A	N/A	N/A	Normal
gnome-session	mohammadyehya	0.00	2850	1.7 MB	2.8 MB	N/A	N/A	N/A	Normal
gnome-session	mohammadyehya	0.00	2857	442.4 kB	20.3 kB	N/A	N/A	N/A	Normal
gnome-session	mohammadyehya	0.00	2939	2.9 MB	7.4 MB	4.1 kB	N/A	N/A	Normal
atspi-bus-daemon	mohammadyehya	0.00	2956	729.1 kB	16.4 kB	N/A	N/A	N/A	Normal
gnome-shell	mohammadyehya	10.38	2958	395.4 MB	1.1 GB	2.2 GB	N/A	N/A	Normal
dbus-daemon	mohammadyehya	0.00	2966	487.4 kB	N/A	N/A	N/A	N/A	Normal
gvfs-udisks2	mohammadyehya	0.00	3013	1.9 MB	194.8 kB	N/A	N/A	N/A	Normal
gvfs-goa-volume-monitor	mohammadyehya	0.00	3023	1.1 MB	143.4 kB	N/A	N/A	N/A	Normal
gvfs-gphoto2-volume-monitor	mohammadyehya	0.00	3028	1.0 MB	98.3 kB	N/A	N/A	N/A	Normal
gvfs-mtp-volume-monitor	mohammadyehya	0.00	3032	383.9 kB	98.3 kB	N/A	N/A	N/A	Normal
gvfs-goa-volume-monitor	mohammadyehya	0.00	3036	536.0 kB	98.3 kB	N/A	N/A	N/A	Normal

## CPU & MEM Graphs (when running CPU intensive programs)



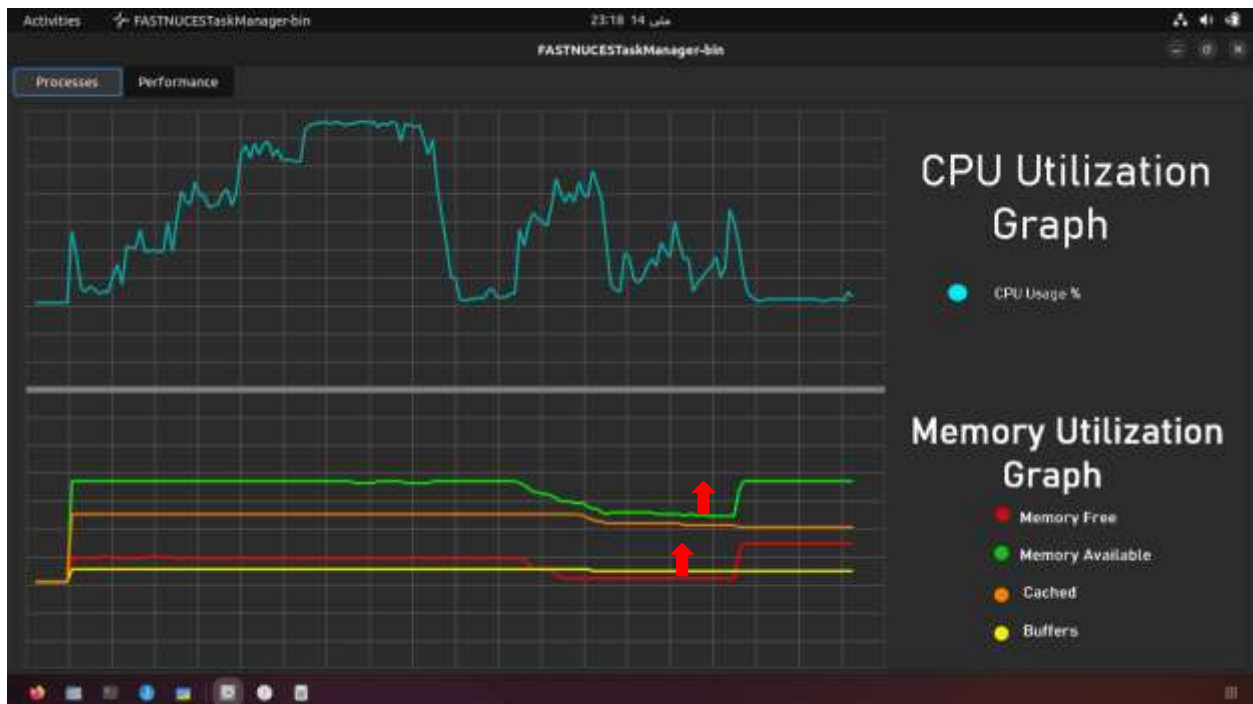
## CPU & MEM Graphs (when killing CPU intensive programs)



## CPU & MEM Graphs (when running MEM intensive programs)



## CPU & MEM Graphs (when killing MEM intensive programs)



## Problems & Challenges

There were many hurdles that we had to face to complete this project. The first problem being that many of GTK+ functions are actually deprecated. This means that no new updates will be further posted, and this also translates to no new bug fixes. Therefore, we had to work around this issue.

Another issue was using C as our language. This meant that we no longer had access to the STL library which included a bunch of useful functions. Therefore, we had to do everything from scratch.

## Conclusion & Breakdown

In the end, the outcome of the project is very satisfying, and we have learnt a lot from participating in this project.

Implemented Basic GUI → Done by 17/Apr/2023

Implemented Basic CPU Graph → Done by 27/Apr/2023

Fixed CPU Graph out of Bounds → Done by 29/Apr/2023

Made Improvements → Done by 1/May/2023

Implemented Basic List → Done by 2/May/2023

Improved CPU Graph → Done by 3/May/2023

Implemented Basic MEM Graph → Done by 3/May/2023

Implemented Grid → Done by 3/May/2023

Completed Graph Section → Done by 4/May/2023

Fixed GtkTreeIter not existing → Done by 7/May/2023

Fixed GtkTreeView & GtkTreeModel issue → Done by 10/May/2023

Implemented List Updating property → Done by 13/May/2023

Fixed Segmentation Fault (bad malloc) → Done by 14/May/2023

# Thank You