

Audit Optimisation Quantité de mémoire vive Bomberman Extended

Nada Mesrati, Anthony Schatt, Quentin Frelat & Morgane Flamant.

Inconvénient avant optimisation

Lors de l'exécution de notre programme nous avons constaté qu'il y avait différents problèmes:

- Le gel des programmes lancés
- La lenteur des déplacements de joueur
- Le décalage entre les différents programmes
- L'obligation de ne lancer qu'à deux joueur pour que ça fonctionne

A cause de ces problèmes nous avons décidé de regarder plus en détail quel optimisation nous pourrions faire pour améliorer ou corriger ces problèmes. Et nous avons trouvé quelques outils pour cela.

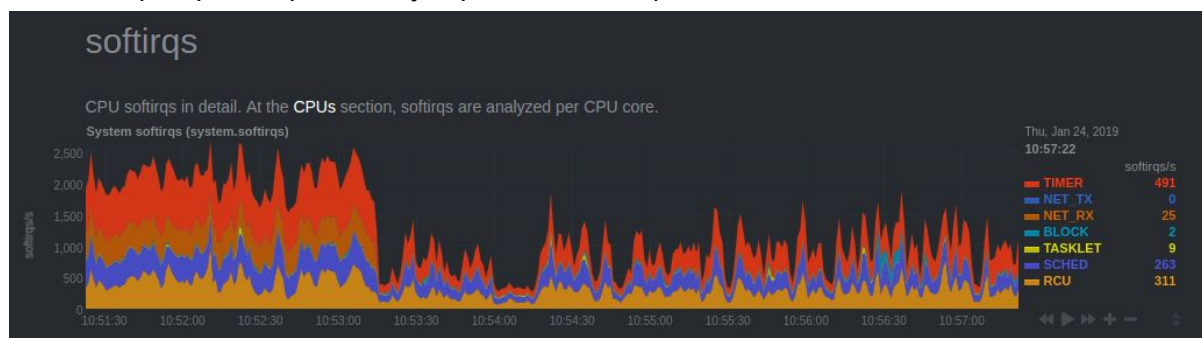
Grâce à l'utilisation de ces outils nous avons décidé d'optimiser la mémoire vive de notre Bomberman. NetData est un utilitaire à installer pour avoir une surveillance des performances de l'ordinateur en temps réel. On retrouve toutes les données que ce soit pour la RAM, le CPU, le Network ou plein d'autres ressources encore. Suite à la réception de ces données on s'est décidé sur l'optimisation de la RAM. Pour mesurer ces données nous avons aussi utilisé le System Monitor de l'ordinateur. Avec ces deux outils combinés nous avons pu voir l'impact de nos modifications.

Ci dessous un extrait de l'utilisation de NetData combiné avec le lancement de notre Bomberman avant les modifications d'optimisation.

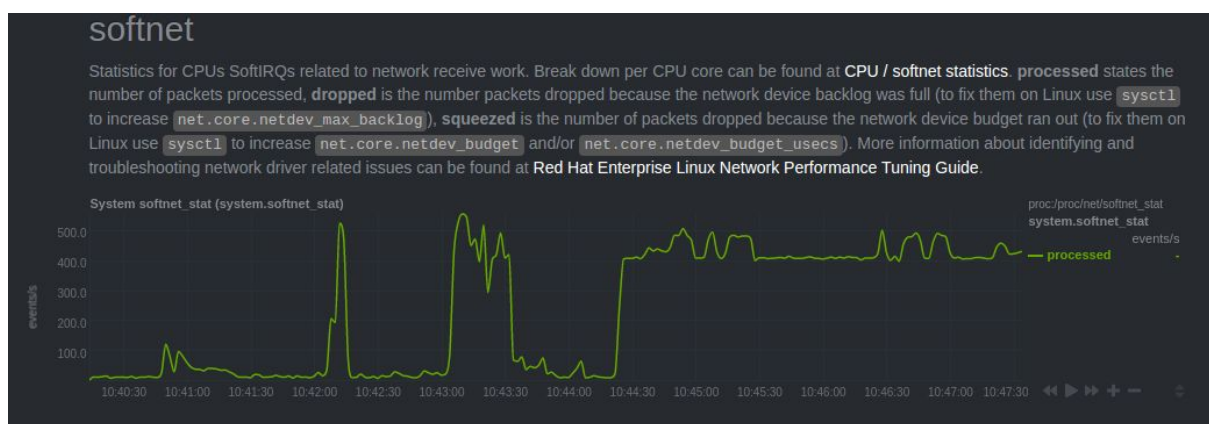
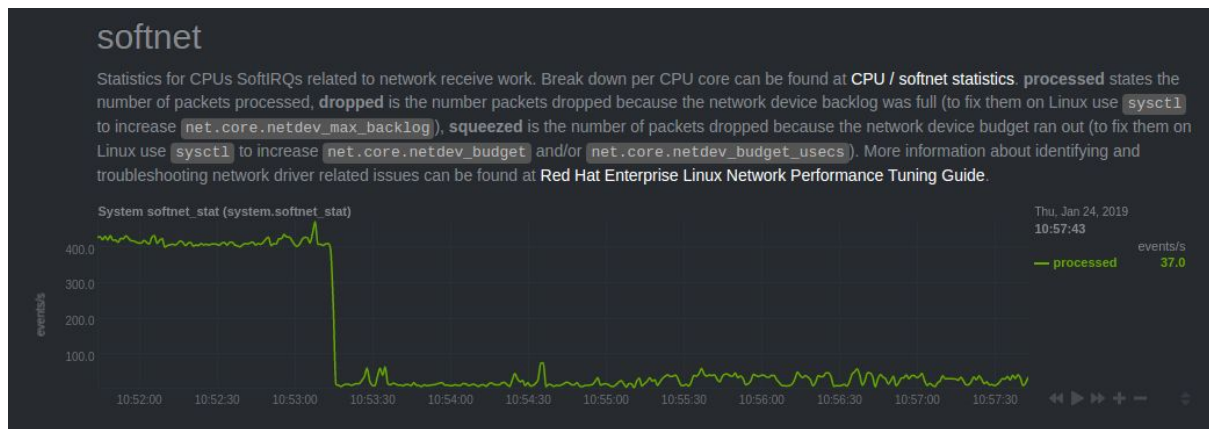
Nous avons regardé les différentes informations que pouvait nous fournir NetData sur notre programme.

Le softirqs de notre ancien Bomberman.

Le système softirq du noyau Linux est un mécanisme permettant d'exécuter du code en dehors du contexte d'un gestionnaire d'interruptions implémenté dans un pilote. On remarque que les pics vont jusqu'à 2500softirqs/s.



Avec le softnet on peut voir le nombre de paquet perdu pendant l'utilisation de notre programme c'est ce qu'on retrouve dans ces deux graphiques:

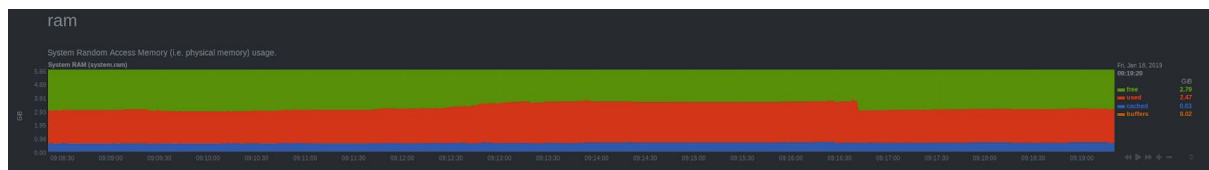


Sur le premier graphique c'est le lancement de notre ancien programme, on voit une perte consécutive de paquet allant à plus de 400events/s et ensuite tout le long du programme on continue de perdre une grande quantité de paquets, ce qui nuit forcément au jeu.

Passons à la RAM:

La mémoire vive est la mémoire informatique dans laquelle peuvent être stockées, puis effacées, les informations traitées par un appareil informatique.

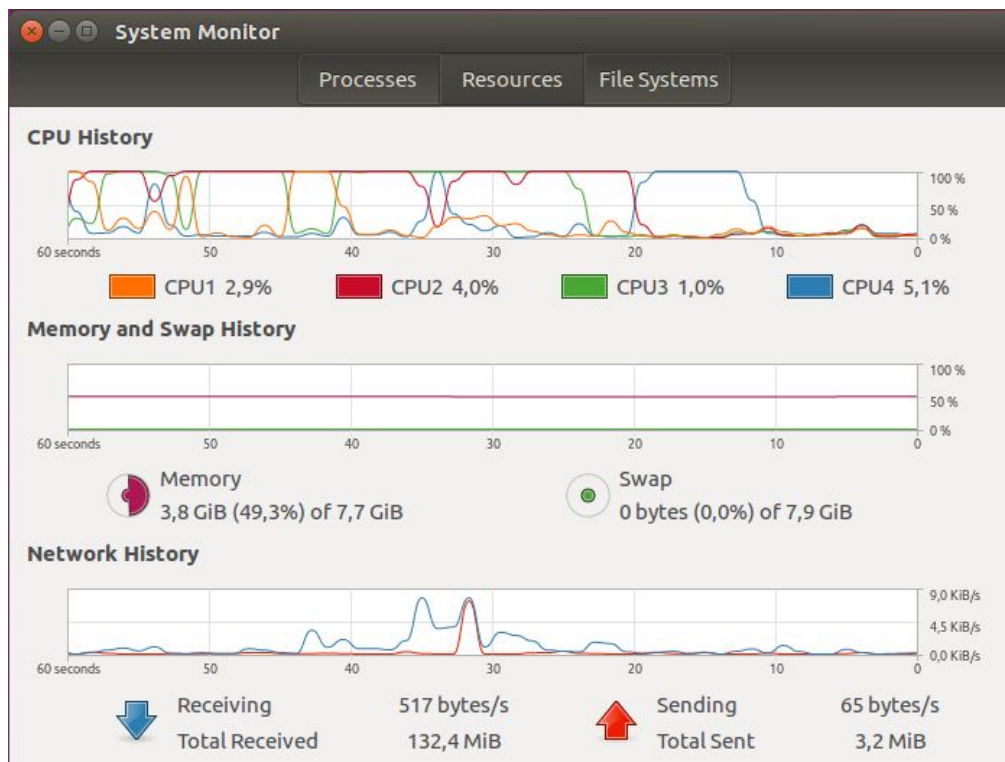
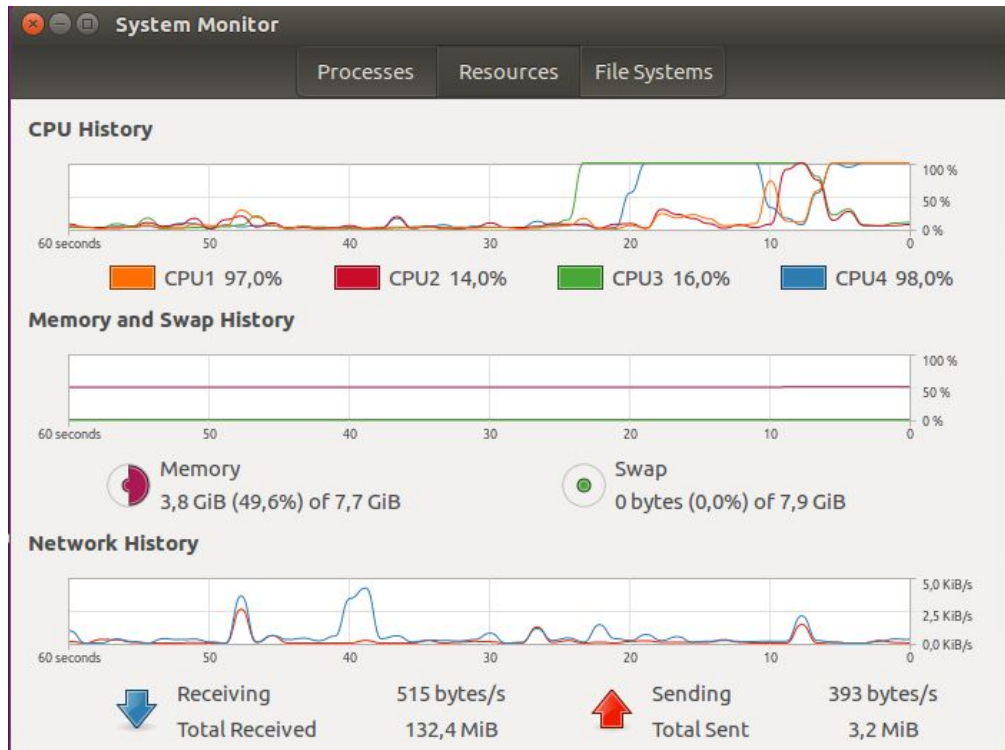
On voit qu'il y a un pic d'utilisation de la RAM, ce qui crée un temps de latence lors du lancement de notre programme. La RAM cherche à allouer de l'espace pour notre programme et cela créer un gel au lancement.



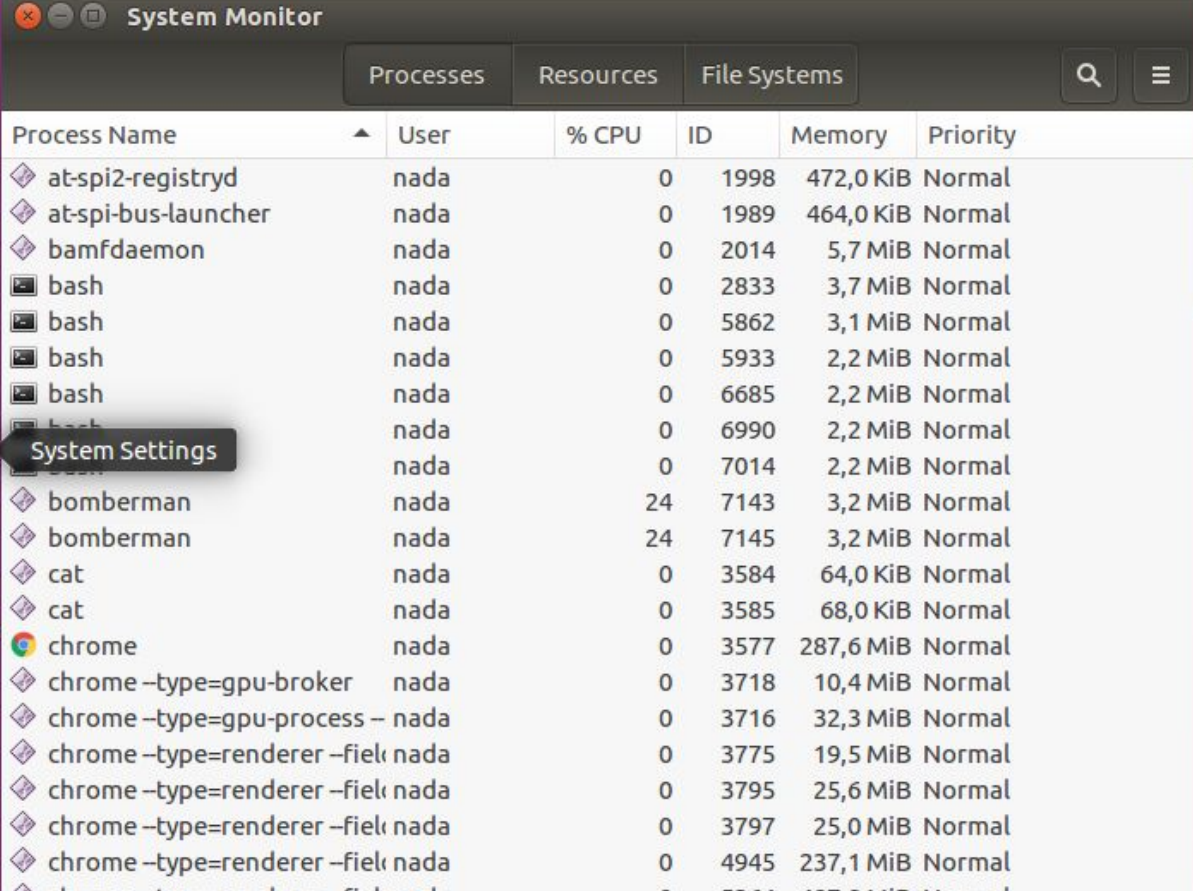
Pour le CPU:

Un processeur est un composant présent dans de nombreux dispositifs électroniques qui exécute les instructions machine des programmes informatiques.

Ci dessous on peut voir le System Monitor de l'ordinateur nous donner des informations sur l'utilisation du CPU, de la mémoire et du Networ



Et enfin sur cette dernière image on peut voir les mêmes données qu'au dessus mais sur les programmes eux-même et leurs consommations. Le bomberman (ici seulement lancé en deux exemplaires) consomme 24% chacun du CPU alors que les autres programmes 0%.



Process Name	User	% CPU	ID	Memory	Priority
at-spi2-registryd	nada	0	1998	472,0 KiB	Normal
at-spi-bus-launcher	nada	0	1989	464,0 KiB	Normal
bamfd daemon	nada	0	2014	5,7 MiB	Normal
bash	nada	0	2833	3,7 MiB	Normal
bash	nada	0	5862	3,1 MiB	Normal
bash	nada	0	5933	2,2 MiB	Normal
bash	nada	0	6685	2,2 MiB	Normal
bash	nada	0	6990	2,2 MiB	Normal
System Settings	nada	0	7014	2,2 MiB	Normal
bomberman	nada	24	7143	3,2 MiB	Normal
bomberman	nada	24	7145	3,2 MiB	Normal
cat	nada	0	3584	64,0 KiB	Normal
cat	nada	0	3585	68,0 KiB	Normal
chrome	nada	0	3577	287,6 MiB	Normal
chrome --type=gpu-broker	nada	0	3718	10,4 MiB	Normal
chrome --type=gpu-process --	nada	0	3716	32,3 MiB	Normal
chrome --type=renderer --fiel	nada	0	3775	19,5 MiB	Normal
chrome --type=renderer --fiel	nada	0	3795	25,6 MiB	Normal
chrome --type=renderer --fiel	nada	0	3797	25,0 MiB	Normal
chrome --type=renderer --fiel	nada	0	4945	237,1 MiB	Normal

Développements réalisés

La première étape fut de supprimer toutes les variables que nous n'utilisons pas, celles-ci prenaient de la place dans la mémoire alors qu'elles ne servaient à rien pour le bon fonctionnement du programme.

```
typedef struct s_bomb
{
    double ticks_left;
    int y;
    int x;
    t_bomb* prev;
    t_bomb* next;
}s_bomb;

typedef struct s_player_info
{
    char connected;
    char alive;
    int x_pos;
    int y_pos;
    int current_dir;
    int current_speed;
    int max_speed;
    int bombs_left;
    int bombs_capacity;
    int frags;
} t_player_info;

typedef struct s_client_request
{
    unsigned int magic; /* Une valeur magique pour vérifier la validité de la requête */
    int x_pos; /* La position x du joueur */
    int y_pos; /* La position y du joueur */
    int dir; /* La direction du joueur */
    int command; /* Une commande valide */
    int speed; /* La vitesse du joueur */
    int checksum; /* Une somme de contrôle pour vérifier la validité de la requête */
} t_client_request;

typedef char t_map[MAP_SIZE];

typedef struct s_game
{
    t_player_info players[MAX_PLAYERS];
    t_map map;
    t_bomb* bomb;
} t_game;

typedef struct s_server {
    pthread_t tid;
    t_game game;
    int fds[MAX_PLAYERS];
    struct sockaddr_in sock_serv;
    struct sockaddr *sock_ptr;
    socklen_t len;
    int sockfd;
    int port;
    int running;
} t_server;
```

```
typedef struct s_bomb
{
    double ticks_left;
    int y;
    int x;
    t_bomb* prev;
    t_bomb* next;
}s_bomb;

typedef struct s_player_info
{
    char connected;
    char alive;
    int x_pos;
    int y_pos;
    int bombs_left;
} t_player_info;

typedef struct s_client_request
{
    int x_pos; /* La position x du joueur */
    int y_pos; /* La position y du joueur */
    int command; /* Une commande valide */
} t_client_request;

typedef char t_map[MAP_SIZE];

typedef struct s_game
{
    t_player_info players[MAX_PLAYERS];
    t_map map;
    t_bomb* bomb;
} t_game;

typedef struct s_server {
    pthread_t tid;
    t_game game;
    int fds[MAX_PLAYERS];
    struct sockaddr_in sock_serv;
    struct sockaddr *sock_ptr;
    socklen_t len;
    int sockfd;
    int port;
    int running;
} t_server;
```

La deuxième étape fut d'utiliser Pragma Pack, cela permet de réduire la taille des structures. Il ordonne au compilateur d'empaqueter les membres de la structure avec un alignement particulier. Il a donc été placé dans les fichiers headers qui contenaient des structures.

Et enfin la dernière étape fut de changer la structure d'une de nos fonctions qui utilisait un malloc à l'intérieur d'une boucle, ce faisant nous allouions de la mémoire en continu dans cette boucle. En déplaçant ce malloc à l'extérieur de la boucle nous avons réduit cette consommation de mémoire vive dans notre programme.

<pre> 54 -void display(SDL_Surface *screen, t_game *game/*, t_player_info *player*/) 55 { -- 56 for (int y = 0; y < MAP_COL; ++y) -- </pre>	<pre> 50 +void display(SDL_Surface *screen, t_game *game/*, t_player_info *player*/) 51 { 52 + SDL_Rect* rect; 53 + SDL_Rect* rect2; 54 + rect = malloc(sizeof(SDL_Rect)); 55 + rect2 = malloc(sizeof(SDL_Rect)); 56 for (int y = 0; y < MAP_COL; ++y) -- </pre>
--	---

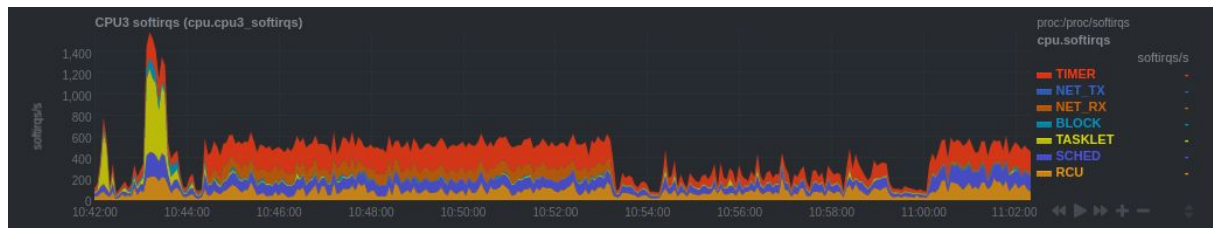
<pre> 1 + #include <SDL_image.h> 2 -#include "Headers/display.h" 3 -#include "Headers/map.h" 4 5 -static SDL_Rect* sdlh_rect(int x, int y, int w, int h) { 6 - SDL_Rect* rect; 7 - 8 - rect = malloc(sizeof(SDL_Rect)); 9 if (!rect) </pre>	<pre> 1 + #include "Headers/bomberman.h" 2 + #include "Headers/client.h" 3 4 +static SDL_Rect* sdlh_rect(SDL_Rect* rect, int x, int y, int w, int h) { 5 if (!rect) </pre>
---	--

Ces trois étapes combinées nous ont permis de réduire considérablement l'utilisation de la mémoire vive de notre programme.

Résultats obtenus

On remarque donc une net amélioration dans notre Bomberman amélioré:

Par rapport au Softirqs:



Cette fois les pics vont maximum jusqu'à 1400softirqs/s. Et si on compare ces deux graphiques on voit que tout le long de l'utilisation du programme les pics sont moins importants et diminuent petit à petit.

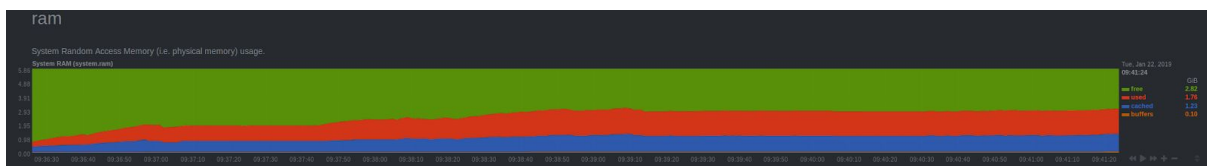
Par rapport au paquets perdus ces graphiques vont nous montrer la différence avec notre programme optimisé:





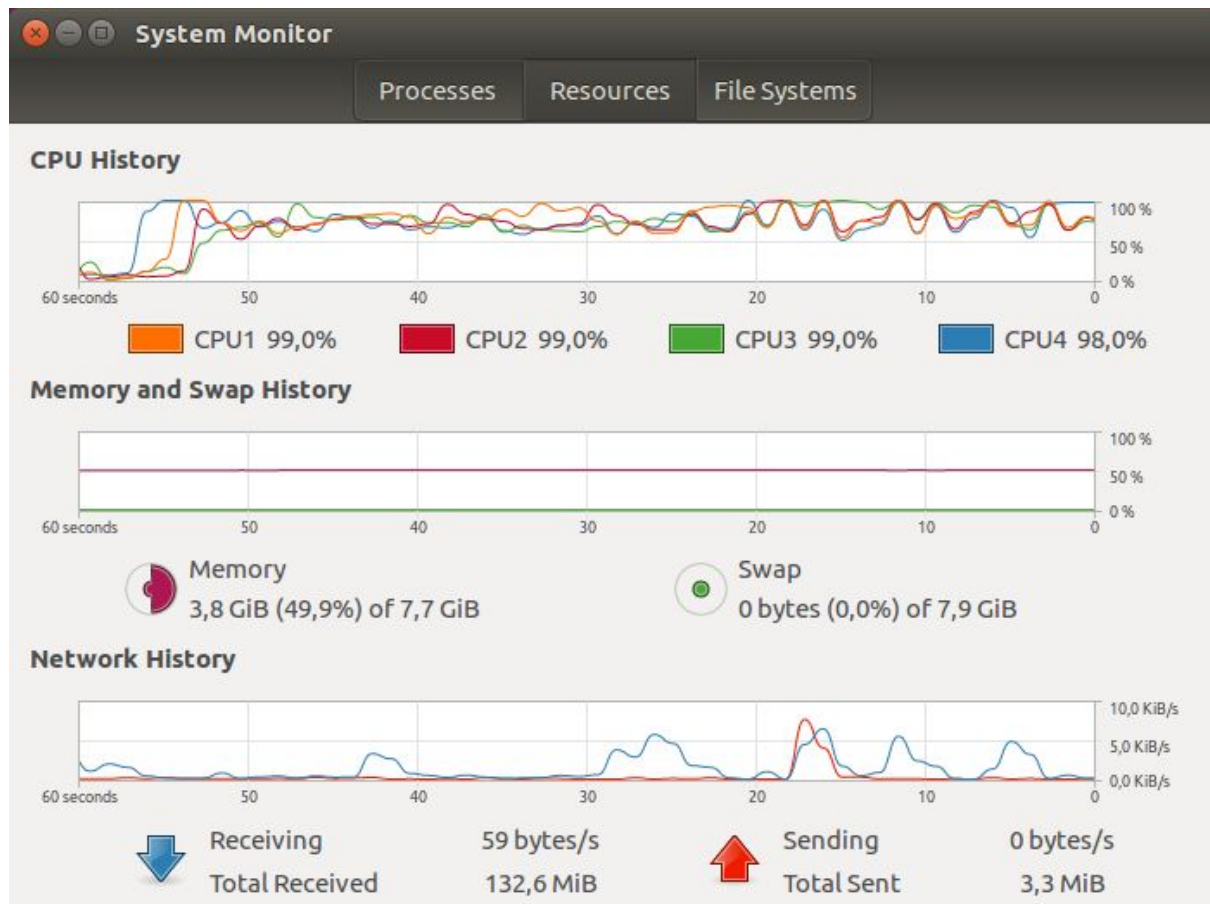
Il y a toujours des paquets qui se perdent mais l'échelle à changer. On ne monte plus à 400 events/s mais à 200 events/s ce qui veut dire que le nombre de paquet perdu a été divisé par 2. Il s'agit d'une grande différence. On peut remarquer aussi que tout le long du programme le nombre de paquet perdu diminue aussi à par quelques pics à des moments spécifiques le reste du temps le nombre de paquet perdu est faible.

Sur cette image de NetData le lancement de notre Bomberman optimisé ne crée pas ce pic de RAM et ne fait pas geler l'ordinateur à son lancement. La RAM est utilisée constamment mais sans créer de discontinuité.



Sur cette image que nous pouvons comparer à celle du dessus, nous avons cette fois quatre programmes de notre Bomberman (Donc le nombre maximum de joueur) et le CPU utilisé pour chacun de ces programmes est inférieur à ce que nous avons pour seulement deux non optimisés.

	bomberman	nada	18	7554	3,2 MiB	Normal
	bomberman	nada	12	7556	3,2 MiB	Normal
	bomberman	nada	13	7559	3,2 MiB	Normal
	bomberman	nada	6	7560	3,2 MiB	Normal



Par rapport à notre ancien programme et les points que nous avons déclarés à problèmes on peut voir des améliorations et des corrections sur ces points:

- Les programmes ne gèlent plus (sauf très faible capacité de l'ordinateur)
- Les déplacements des joueurs sont plus rapides
- Les programmes n'ont plus de décalage entre eux
- Le programme se lance correctement à quatre joueurs