

## **Programmation système en C sur Unix/Linux**

**Durée:** 3 jours

Prix et dates: nous consulter

### **Public:**

Toute personne amenée à programmer, à superviser ou à modifier des logiciels écrits en langage C et liés au système d'exploitation.

### **Objectifs:**

Compléter des connaissances en langage C par une formation approfondie sur les mécanismes d'accès au système d'exploitation. L'accent sera particulièrement sur les fichiers, pointeurs, allocations de mémoire, communications et les bibliothèques systèmes.

### **Connaissances préalables nécessaires:**

Il est demandé aux participants de bien connaître les structures et fonctions de base du langage C.

### **Programme:**

- Rappels** : Architecture d'un programme écrit en C.  
Phases de compilation.
- Gestion de la mémoire** : Rappel sur l'organisation de la mémoire.  
L'adressage par les pointeurs.  
Les opérateurs `&` et `*`.  
Les pointeurs et les arguments de fonctions.  
Les calculs d'adresses.  
Les fonctions d'allocation `malloc` et `free`,  
et les appels systèmes: `sbrk`, `realloc`.  
Travaux pratiques :  
écriture d'un allocateur de mémoire.

## **Programmation système en C sur Unix/Linux**

**Communications inter-processus.**

: Les différentes méthodes : pipes, fifo, signaux, files de messages.  
Signaux et interruptions : les principaux signaux.  
Travaux pratiques :  
émission d'un signal avec `kill()`, réception du signal par `signal()`.  
Sémaphores et appels concurrents :  
principe de fonctionnement des sémaphores.  
Travaux pratiques :  
mise en oeuvre avec `semget`, `semctl`, `semop`.  
Segments de mémoires partagées :  
définitions de constantes et structures,  
Travaux pratiques :  
création d'un segment de mémoire partagée avec `shmget`,  
attachement, détachement d'un segment avec `shmat`, `shmdt`.  
Files de messages :  
constantes et structures nécessaires pour la manipulation des  
files de messages.  
Travaux pratiques :  
mise en oeuvre de la primitive `msgget()`,  
gestion des files de messages (consultation, modification,  
suppression) avec `msgctl()`  
Envoi d'un message à une file : `msgsend()`.  
Segments partagés :  
définition d'un segment de mémoire partagé. Description et  
mise en oeuvre des appels systèmes `shmat()`, `shmget()`.  
Utilisation de sémaphores pour la gestion des accès  
concurrents au segment.  
Sockets BSD :  
mise en oeuvre des prises réseaux pour la communication  
interprocessus.  
Exemple avec des liens locaux. Extension aux liens distants.  
Communications inter-machines.

## **Programmation système en C sur Unix/Linux**

### **Les processus et la parallélisation**

: Création de processus.  
Définition et mise en oeuvre des primitives `fork()`, `clone()`, `setsid()`.  
Limites d'utilisation. Introduction aux threads.  
Les threads. La norme et les implémentations.  
L'implémentation Posix : NPTL.  
Cycle de vie des threads: création, destruction.  
Synchronisation entre threads, détachement du processus principal, attente de fin d'exécution.  
Attributs des threads.  
Gestion de la mémoire consommée, gestion de la pile de données.  
Gestion des accès concurrents, principe de l'exclusion mutuelle.  
Travaux pratiques :  
mise en oeuvre des mutex.  
Coopération de traitements entre threads.  
Mise en oeuvre des conditions variables. Gestion des signaux dans un thread.  
Ordonnancement de threads.