



## Formation debugging, profiling, tracing et analyse de performance sous Linux

Durée de la formation

⌚ 4 demi-journées – 16 h

Langue

Transparents      Anglais

Présentation      Français  
Anglais  
Italien

Formateur

Un des ingénieurs suivants

- Alexis Lothoré
- Luca Ceresoli
- Olivier Benjamin

Contact

✉️ [training@bootlin.com](mailto:training@bootlin.com)  
📞 +33 4 84 25 80 96

### Public visé

Sociétés et ingénieurs intéressés dans le debug, profiling et tracing de systèmes et d'applications Linux, afin d'analyser et résoudre des problèmes de performance ou de latence.

### Objectifs opérationnels

- Être capable de comprendre les principaux concepts de Linux qui sont liés à l'analyse de performance : processus, threads, gestion de la mémoire, mémoire virtuelle, contextes d'exécution, etc.
- Être capable d'analyser pourquoi un système est chargé et quels sont les éléments qui contribuent à cette charge avec les outils usuels d'observabilité sous Linux.
- Être capable de débugger une application espace utilisateur avec *gdb*, soit en direct soit *post-mortem* suite à un crash, et analyser le contenu de binaires ELF.
- Être capable d'utiliser le *tracing* et le *profiling* sur une application espace utilisateur et comprendre ses interactions avec le noyau Linux afin de corriger des bugs, en utilisant *strace*, *ltrace*, *perf* ou *Callgrind*
- Être capable d'utiliser le *tracing* et le *profiling* le système Linux complet, en utilisant *perf*, *ftrace*, *kprobe*, les outils *eBPF*, *kernelshark* ou *LTTng*
- Être capable de débugger des problèmes au niveau du noyau Linux : debug de crash en direct ou post-mortem, analyse de problèmes mémoire au niveau noyau, analyse de problèmes de locks, utilisation de debuggers au niveau noyau.

### Prérequis

- **Connaissance et pratique des commandes UNIX ou GNU/Linux** : les participants doivent être à l'aise avec l'utilisation de la ligne de commande Linux. Les participants manquant d'expérience sur ce sujet doivent se former par eux-mêmes, par exemple en utilisant nos supports de formation.
- **Expérience minimale en développement Linux embarqué** : les participants doivent avoir une compréhension minimale de l'architecture d'un système Linux embarqué : rôle du noyau Linux par rapport à l'espace utilisateur, développement d'applications espace utilisateur en C. Suivre la formation Linux embarqué de Bootlin permet de remplir ce pré-requis.
- **Niveau minimal requis en anglais** : B1, d'après le *Common European Framework of References for Languages*, pour nos sessions animées en anglais. Voir la grille CEFR pour une auto-évaluation.

### Méthodes pédagogiques

- Présentations animées par le formateur, par visioconférence. Les participants peuvent poser des questions à tout instant.
- Démonstrations pratiques réalisées par le formateur, basés sur les travaux pratiques de la formation, par vidéo-conférence. Les participants peuvent poser des questions à tout instant. Optionnellement, les participants qui ont accès aux accessoires matériels de la formation peuvent reproduire par eux-même les travaux pratiques.
- Messagerie instantanée pour questions entre les sessions (réponse sous 24h, hors week-end et jours fériés)
- Version électronique des supports de présentation, des instructions et des données de travaux pratiques. Les supports sont librement disponibles [ici](#).

### Modalités d'évaluation

Seuls les participants qui auront assisté à l'intégralité des journées de formation, et qui auront obtenu plus de 50% de réponses correctes à l'évaluation finale recevront une attestation individuelle de formation de la part de Bootlin.

### Handicap

Les participants en situation de handicap qui ont des besoins spécifiques sont invités à nous contacter à l'adresse [training@bootlin.com](mailto:training@bootlin.com) afin de discuter des adaptations nécessaires à la formation.

## Équipement nécessaire

Équipement obligatoire :

- Ordinateur avec le système d'exploitation de votre choix, équipé du navigateur Google Chrome ou Chromium pour la conférence vidéo.
- Une webcam et un micro (de préférence un casque avec micro)
- Une connexion à Internet à haut débit

Optionnellement, si les participants souhaitant pouvoir reproduire par eux-mêmes les travaux pratiques, ils doivent acheter séparément la carte de développement et les accessoires associés, et devront disposer d'un PC avec une installation native d'Ubuntu Linux 24.04.

## Plateforme matérielle pour les travaux pratiques

### Plateforme STM32MP1

Une de ces cartes de STMicroelectronics :

**STM32MP157A-DK1**, **STM32MP157D-DK1**, **STM32MP157C-DK2** ou **STM32MP157F-DK2**

- Processeur STM32MP157, double Cortex-A7, de STMicroelectronics
- Alimentée par USB
- 512 Mo DDR3L RAM
- Port Gigabit Ethernet port
- 4 ports hôte USB 2.0
- 1 port USB-C OTG
- 1 connecteur Micro SD
- Debugger ST-LINK/V2-1 sur la carte
- Connecteurs compatibles Arduino Uno v3
- Codec audio
- Divers : boutons, LEDs
- Écran LCD tactile (uniquement sur cartes DK2)



### BeaglePlay

Carte BeaglePlay

- SoC Texas Instruments AM625x (CPU 4xARM Cortex-A53)
- SoC avec accélération 3D, MCU intégré et de nombreux autres périphériques.
- 2 GB de RAM
- 16 Go de stockage eMMC
- USB hôte et device, microSD, HDMI
- WiFi 2.4 and 5 GHz, Bluetooth et aussi Ethernet
- 1 Header MicroBus (SPI, I2C, UART, ...), connecteurs OLEI et CSI.



## Programme de la formation

### Demi-journée 1

Cours	Pile logicielle Linux	<ul style="list-style-type: none"><li>▪ Vue d'ensemble : comprendre l'architecture général d'un système Linux, aperçu des principaux composants</li><li>▪ Différence entre un processus et un thread, comment les applications fonctionnent de façon concurrente.</li><li>▪ Fichiers ELF et outils d'analyse associés.</li><li>▪ Organisation de l'espace d'adressage des applications : heap, stack, bibliothèques partagées, etc.</li><li>▪ MMU et gestion mémoire : espaces d'adressage physique et virtuel</li><li>▪ Contexte d'exécution dans le noyau : threads noyau, workqueues, interruptions, interruptions threadées, softirq</li></ul>
Cours	Outils usuels d'analyse et d'observation	<ul style="list-style-type: none"><li>▪ Analyse d'un binaire ELF avec les outils GNU (<i>objdump</i>, <i>addr2line</i>)</li><li>▪ Outils pour monitorer un système Linux : processus, consommation et mapping mémoire, ressources</li><li>▪ Utilisation de <i>vmstat</i>, <i>iostat</i>, <i>ps</i>, <i>top</i>, <i>iotop</i>, <i>free</i> et compréhension des métriques qu'ils fournissent.</li><li>▪ Systèmes de fichiers virtuels : <i>procfs</i>, <i>sysfs</i> et <i>debugfs</i></li></ul>
Démonstration	Comprendre ce qui fonctionne sur un système et sa charge	<ul style="list-style-type: none"><li>▪ Observation des processus en cours d'exécution avec <i>ps</i> et <i>top</i></li><li>▪ Observation des mappings mémoire avec <i>procfs</i> et <i>pmap</i></li><li>▪ Monitoring d'autres ressources avec <i>iostat</i>, <i>vmstat</i> et <i>netstat</i></li></ul>
Cours	Debug d'une application	<ul style="list-style-type: none"><li>▪ Utilisation de <i>gdb</i> sur un processus en cours d'exécution.</li><li>▪ Comprendre l'impact des optimisations du compilateur sur la capacité à débugger un programme.</li><li>▪ Analyse post-mortem avec des fichiers <i>core</i></li><li>▪ Debug à distance avec <i>gdbserver</i>.</li><li>▪ Étendre les capacités de <i>gdb</i> en utilisant des scripts Python.</li></ul>

### Demi-journée 2

Démonstration	Résoudre un crash applicatif	<ul style="list-style-type: none"><li>▪ Analyse d'un code C compilé avec <i>compiler-explorer</i> pour comprendre les optimisations.</li><li>▪ Utilisation de <i>gdb</i> en ligne de commande, puis depuis un IDE.</li><li>▪ Utilisation des possibilités de scripting Python dans <i>gdb</i>.</li><li>▪ Debugger une application <i>post mortem</i> avec un <i>core dump</i> et <i>gdb</i></li></ul>
Cours	Tracing d'une application	<ul style="list-style-type: none"><li>▪ Tracing des appels systèmes avec <i>strace</i>.</li><li>▪ Tracing des appels à des bibliothèques partagées avec <i>ltrace</i>.</li></ul>
Démonstration	Débugger des problèmes applicatifs	<ul style="list-style-type: none"><li>▪ Analyser les appels à des bibliothèques partagées d'une application en utilisant <i>ltrace</i>.</li><li>▪ Débugger une application qui fonctionne de manière incorrecte en utilisant <i>strace</i>.</li></ul>
Cours	Problèmes liés à la mémoire	<ul style="list-style-type: none"><li>▪ Problèmes classiques liés à la mémoire : <i>buffer overflow</i>, <i>segmentation fault</i>, fuite mémoire, collision pile/tas.</li><li>▪ Outils de détection/investigation de problèmes mémoires : <i>valgrind</i>, <i>libfence</i>, etc.</li><li>▪ Profiling de l'utilisation du tas en utilisant <i>Massif</i></li></ul>

Démo	Debugger des problèmes liés à la mémoire	<ul style="list-style-type: none"> <li>▪ Fuites mémoire et détection de comportement incorrects avec <i>valgrind</i> et <i>vgdb</i>.</li> <li>▪ Problèmes de performance liés à une sur-allocation.</li> <li>▪ Visualisation de l'utilisation du tas par une application en utilisant <i>Massif</i>.</li> </ul>
------	--	---

### Demi-journée 3

Cours	Profiling d'application	<ul style="list-style-type: none"> <li>▪ Problèmes de performance.</li> <li>▪ Récupération d'informations de profiling avec <i>perf</i>.</li> <li>▪ Analyse du graphe d'appel d'une application avec <i>Callgrind</i> et <i>KCacheGrind</i>.</li> <li>▪ Filtrage du jeu de données récupéré.</li> <li>▪ Interprétation, des données enregistrées avec <i>perf</i>.</li> </ul>
Démo	Profiling d'application	<ul style="list-style-type: none"> <li>▪ Profiling d'une application avec <i>Callgrind/KCacheGrind</i>.</li> <li>▪ Analyse des performances d'une application avec <i>perf</i>.</li> <li>▪ Générer un <i>flamegraph</i> avec <i>FlameGraph</i>.</li> </ul>
Cours	Profiling et tracing de l'ensemble du système	<ul style="list-style-type: none"> <li>▪ Profiling du système complet avec <i>perf</i>.</li> <li>▪ Utilisation de <i>kprobes</i> pour ajouter des points de trace supplémentaires sans recompilation</li> <li>▪ Tracing d'application et du noyau et visualisation des traces avec <i>ftrace</i>, <i>kernelshark</i> ou <i>LTTng</i></li> <li>▪ Tracing avec eBPF : principes généraux, développements d'outils avec BCC puis libbpf</li> </ul>
Démo	Profiling et tracing de l'ensemble du système	<ul style="list-style-type: none"> <li>▪ Profiling du système complet avec <i>perf</i>.</li> <li>▪ Analyse de latences système avec <i>ftrace</i> et <i>kernelshark</i>.</li> </ul>

### Demi-journée 4

Démo	Développement d'outils en eBPF	<ul style="list-style-type: none"> <li>▪ Scripting python avec <i>BCC</i>.</li> <li>▪ Développement d'un outil personnalisé de tracing avec <i>libbpf</i>.</li> </ul>
Cours	Debugging du noyau Linux	<ul style="list-style-type: none"> <li>▪ Sorties de la compilation du noyau Linux utiles pour le debugging (<i>vmlinux</i>, <i>System.map</i>).</li> <li>▪ Comprendre et configurer le comportement des <i>kernel oops</i>.</li> <li>▪ Analyse post-mortem d'un crash kernel avec <i>crash</i>.</li> <li>▪ Problèmes mémoire au niveau kernel (<i>KASAN</i>, <i>UBSAN</i>, <i>Kmemleak</i>).</li> <li>▪ Debugging du noyau Linux avec <i>KGDB</i> et <i>KDB</i>.</li> <li>▪ Options du noyau Linux pour le debug des problèmes de verrous (<i>lockdep</i>)</li> <li>▪ Autres options de configuration du noyau Linux utiles pour le debug.</li> </ul>
Démo	Debugging du noyau Linux	<ul style="list-style-type: none"> <li>▪ Analyse d'un <i>oops</i> après utilisation d'un module noyau incorrect, avec <i>objdump</i> et <i>addr2line</i>.</li> <li>▪ Debugging d'un <i>deadlock</i> avec les options <i>PROVE_LOCKING</i>.</li> <li>▪ Détecter un <i>undefined behavior</i> avec <i>UBSAN</i> dans le noyau Linux.</li> <li>▪ Trouver une fuite mémoire avec <i>kmemleak</i>.</li> <li>▪ Débugger un module noyau avec <i>KGDB</i>.</li> </ul>