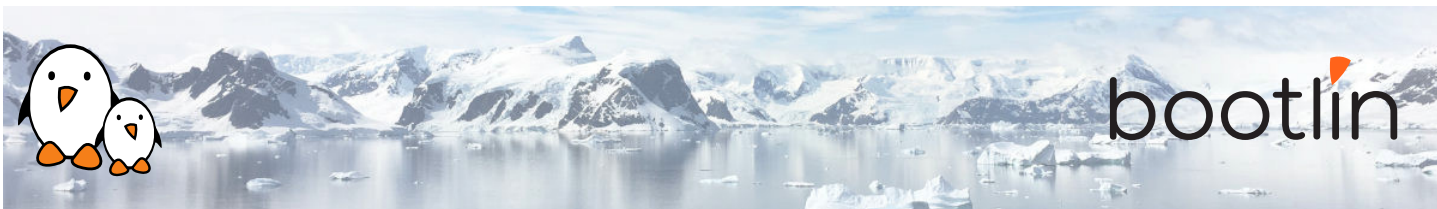


Formation debugging, profiling, tracing et analyse de performance sous Linux

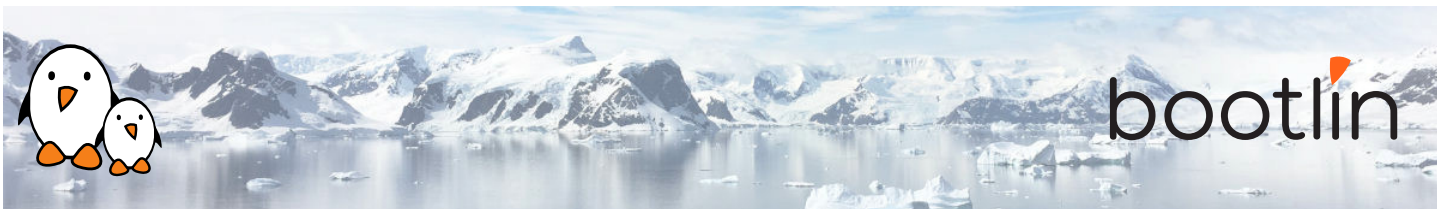
Séminaire en ligne, 4 sessions de 4 hours

Dernière mise à jour : 20 May 2024

Titre	Formation debugging, profiling, tracing et analyse de performance sous Linux
Objectifs opérationnels	<ul style="list-style-type: none">• 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 déboguer une application espace utilisateur avec <i>gdb</i>, soit en direct soit <i>post-mortem</i> suite à un crash, et analyser le contenu de binaires ELF.• Être capable d'utiliser le <i>tracing</i> et le <i>profiling</i> sur une application espace utilisateur et comprendre ses interactions avec le noyau Linux afin de corriger des bugs, en utilisant <i>strace</i>, <i>ltrace</i>, <i>perf</i> ou <i>Callgrind</i>• Être capable d'utiliser le <i>tracing</i> et le <i>profiling</i> le système Linux complet, en utilisant <i>perf</i>, <i>ftrace</i>, <i>kprobe</i>, les outils <i>eBPF</i>, <i>kernelshark</i> ou <i>LTTng</i>• Être capable de déboguer 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.
Durée	Quatre demi-journées - 16 h (4 h par demi-journée)



Méthodes pédagogiques	<ul style="list-style-type: none">• 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 sur https://bootlin.com/doc/training/debugging.
Formateur	Un des ingénieurs mentionnés sur : https://bootlin.com/training/trainers/
Langue	Présentations : Français Supports : Anglais
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.
Pré-requis	<ul style="list-style-type: none">• 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 disponible à l'adresse bootlin.com/blog/command-line/.• 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 <i>Linux embarqué</i> de Bootlin, disponible sur bootlin.com/training/embedded-linux/, permet de remplir ce pré-requis.• Niveau minimal requis en anglais : B1, d'après le <i>Common European Framework of References for Languages</i>, pour nos sessions animées en anglais. Voir bootlin.com/pub/training/cefr-grid.pdf pour une auto-évaluation.



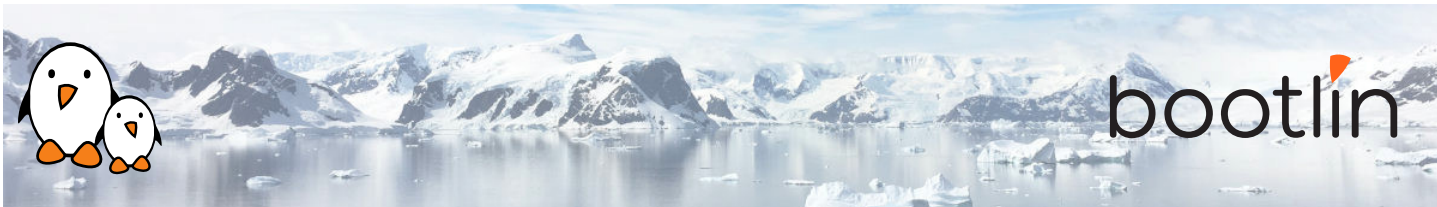
Équipement nécessaire	<ul style="list-style-type: none">• Ordinateur avec le système d'exploitation de votre choix, équipé du navigateur Google Chrome ou Chromium pour la conférence vidéo.• Une webcam et un micro (de préférence un casque avec micro)• Une connexion à Internet à haut débit
Modalités d'évaluation	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 afin de discuter des adaptations nécessaires à la formation.

Matériel utilisé pour les démonstrations pratiques

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)





1^{ère} demi-journée

Cours - Pile logicielle Linux

- Vue d'ensemble : comprendre l'architecture général d'un système Linux, aperçu des principaux composants
- Différence entre un processus et un thread, comment les applications fonctionnent de façon concurrente.
- Fichiers ELF et outils d'analyse associés.
- Organisation de l'espace d'adressage des applications : heap, stack, bibliothèques partagées, etc.
- MMU et gestion mémoire : espaces d'adressage physique et virtuel
- Contexte d'exécution dans le noyau : threads noyau, workqueues, interruptions, interruptions threadées, softirq

Cours - Outils usuels d'analyse et d'observation

- Analyse d'un binaire ELF avec les outils GNU (*objdump*, *addr2line*)
- Outils pour monitorer un système Linux : processus, consommation et mapping mémoire, ressources
- Utilisation de *vmstat*, *iostat*, *ps*, *top*, *iotop*, *free* et compréhension des métriques qu'ils fournissent.
- Systèmes de fichiers virtuels : *procfs*, *sysfs* et *debugfs*

Démo - Comprendre ce qui fonctionne sur un système et sa charge

- Observation des processus en cours d'exécution avec *ps* et *top*
- Observation des mappings mémoire avec *procfs* et *pmap*
- Monitoring d'autres ressources avec *iostat*, *vmstat* et *netstat*

Cours - Debug d'une application

- Utilisation de *gdb* sur un processus en cours d'exécution.
- Comprendre l'impact des optimisations du compilateur sur la capacité à débbugger un programme.
- Analyse post-mortem avec des fichiers *core*
- Debug à distance avec *gdbserver*.
- Étendre les capacités de *gdb* en utilisant des scripts Python.



2^{ème} demi-journée

Démo - Résoudre un crash applicatif

- Analyse d'un code C compilé avec `compiler-explorer` pour comprendre les optimisations.
- Utilisation de `gdb` en ligne de commande, puis depuis un IDE.
- Utilisation des possibilités de scripting Python dans `gdb`.
- Debugger une application *post mortem* avec un `core dump` et `gdb`

Cours - Tracing d'une application

- Tracing des appels systèmes avec `strace`.
- Tracing des appels à des bibliothèques partagées avec `ltrace`.

Démo – Débugger des problèmes applicatifs

- Analyser les appels à des bibliothèques partagées d'une application en utilisant `ltrace`.
- Débugger une application qui fonctionne de manière incorrecte en utilisant `strace`.

Cours - Problèmes liés à la mémoire

- Problèmes classiques liés à la mémoire : `buffer overflow`, `segmentation fault`, fuite mémoire, collision pile/tas.
- Outils de détection/investigation de problèmes mémoires : `valgrind`, `libefence`, etc.
- Profiling de l'utilisation du tas en utilisant `Massif`

Démo – Débugger des problèmes liés à la mémoire

- Fuites mémoire et détection de comportement incorrects avec `valgrind` et `vgdb`.
- Problèmes de performance liés à une sur-allocation.
- Visualisation de l'utilisation du tas par une application en utilisant `Massif`.



3^{ème} demi-journée

Cours – Profiling d’application

- Problèmes de performance.
- Récupération d’informations de profiling avec *perf*.
- Analyse du graphe d’appel d’une application avec *Callgrind* et *KCachegrind*.
- Filtrage du jeu de données récupéré.
- Interprétation, des données enregistrées avec *perf*.

Démo - Profiling d’application

- Profiling d’une application avec *Callgrind/KCachegrind*.
- Analyse des performances d’une application avec *perf*.
- Générer un *flamegraph* avec *FlameGraph*.

Cours - Profiling et tracing de l’ensemble du système

- Profiling du système complet avec *perf*.
- Utilisation de *kprobes* pour ajouter des points de trace supplémentaires sans recompilation
- Outils *eBPF* (*bcctools*, *bpfftrace*, etc) pour les scénarios de tracing complexes.
- Tracing d’application et du noyau et visualisation des traces avec *ftrace*, *kernelshark* ou *LTTng*

4^{ème} demi-journée

Démo - Profiling et tracing de l’ensemble du système

- Profiling du système complet avec *perf*.
- Latence d’interruptions avec *ftrace*.
- Tracing et visualisation de l’activité du système avec *kernelshark* ou *LTTng*



Cours - Debugging du noyau Linux

- Sorties de la compilation du noyau Linux utiles pour le debugging (`vmlinux`, `System.map`).
- Comprendre et configurer le comportement des *kernel oops*.
- Analyse post-mortem d'un crash kernel avec *crash*.
- Problèmes mémoire au niveau kernel (*KASAN*, *UBSAN*, *Kmemleak*).
- Debugging du noyau Linux avec *KGDB* et *KDB*.
- Options du noyau Linux pour le debug des problèmes de verrous (*lockdep*)
- Autres options de configuration du noyau Linux utiles pour le debug.

Démo - Debugging du noyau Linux

- Analyse d'un *oops* après utilisation d'un module noyau incorrect, avec *objdump* et *addr2line*.
- Debugging d'un *deadlock* avec les options *PROVE_LOCKING*.
- Détecter un *undefined behavior* avec *UBSAN* dans le noyau Linux.
- Trouver une fuite mémoire avec *kmemleak*.
- Débugger un module noyau avec *KGDB*.

Temps supplémentaire possible

Du temps supplémentaire (jusqu'à 4 heures) pourrait être proposé si le programme ne tenait pas en 4 demi-journées, selon le temps passé à répondre aux questions des participants.