SOLID開発プラットフォーム QEMU対応サンプル ソースコードの使い方

2021.08.01

京都マイクロコンピュータ株式会社

注意

- ここで説明しているアプリケーションは、あくまでもSOLIDの機能を試していただく為のサンプルとして実装されたものです。
- 厳密なテストは行っておらず、性能のチューニングもされていませんので、動作保証の対象外になります。

- また、一部のソースコード（SOLIDコアサービス実装など）は、ビルド済のライブラリになっております。本ソースコードについては、SOLIDサブスクリプション契約の他に、ソースコード開示契約が必要になります（費用は無償）。

ソースコードアーカイブ

- 32bit ARM用サンプル
  - `solid-os-b9fcb79-aarch32-qemu-20210730.zip`
- 64bit ARM用サンプル
  - `solid-os-b9fcb79-aarch64-qemu-20210730.zip`

添付されているサンプルアプリ

- タスク切り替えサンプル
  - qemu-virtmini-sample-asp3-demo-task-*-S
  - qemu-virtmini-sample-fmp-2CPU-demo-task-*-S
- SOLID独自形式ローダブルサンプル
  - qemu-virtmini-sample-asp3-demo-loader-*-S
- DLL形式ローダブルサンプル
  - qemu-virtmini-sample-asp3-demo-dll-*-S

＊は、32bit用は aarch32 、64bit用は aarch64 となります。

開発環境の準備

- 以下の二つを必須インストールしてください
  - SOLID-2.1.0.exe　　※2.1.0以降であれば利用可能
  - SOLID-QEMU-1.0.0.exe　※1.0.0以降であれば利用可能
- 64bit ARM命令でビルド、デバッグするには、以下を追加でインストールする必要があります
  - SOLID-AArch64-2.1.0.exe　　※2.1.0以降であれば利用可能

- インストール、セットアップについては、以下のURLをご確認ください
  - http://solid.kmckk.com/doc/skit/current/tutorial.html

新規のソリューションの作成

サンプルソリューションを元に、新規のSOLID-OSを含むソリューションを作成する方法を説明します

新規ソリューション作成手順

- SOLID-IDEのメニューにある[ファイル(F)]-[新規作成(N)]-[プロジェクト(P)]は、スタティックライブラリやローダブルアプリなどのためのプロジェクトを作成します
- SOLID-OSを含むソリューションは次の手順で作成してください

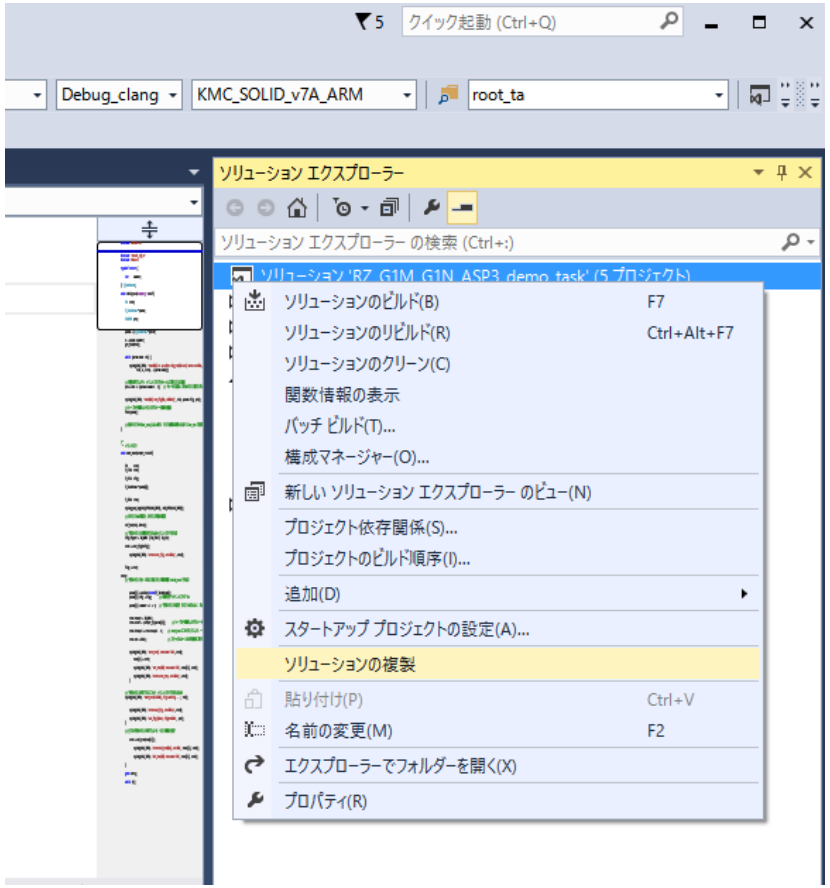
既存のソリューションをSOLID-IDE で開き、「ソリューションの複製」を行う

その際、新しいソリューション名、プロジェクト名を指定

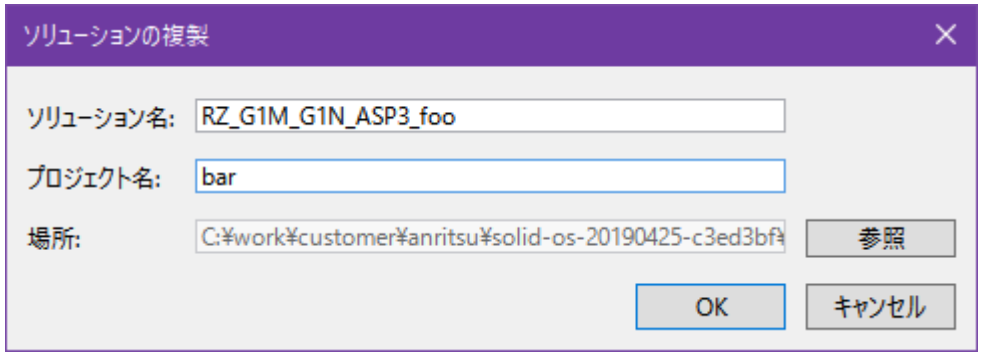
SOLID-IDEがソリューションを複写し、ソリューションファイルとプロジェクトファイルの書き換えを行います

ユーザ様がアプリケーションのコードを書き換える

使い方



ソリューションエクスプローラで、ソリューションの右クリックメニューを開き、”ソリューションの複製”を選択します



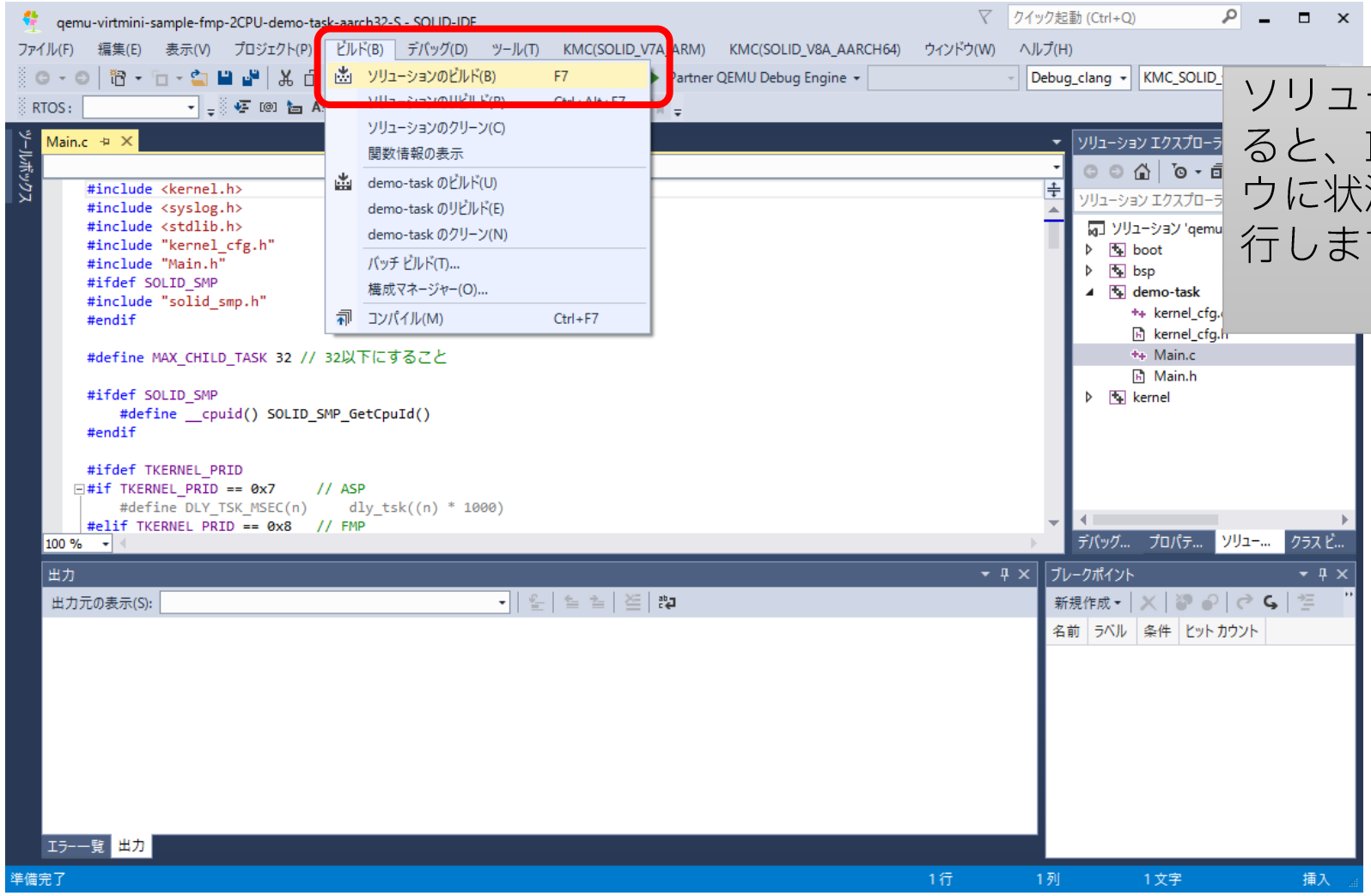
上記ダイアログが出るので、新しいソリューション名と、新しいスタートアッププロジェクト名を入力し、OKをします。
（場所、の変更はしないでください）
完了後、新規のソリューションが自動的に開かれます

タスク切り替えサンプル

タスク切り替えサンプル

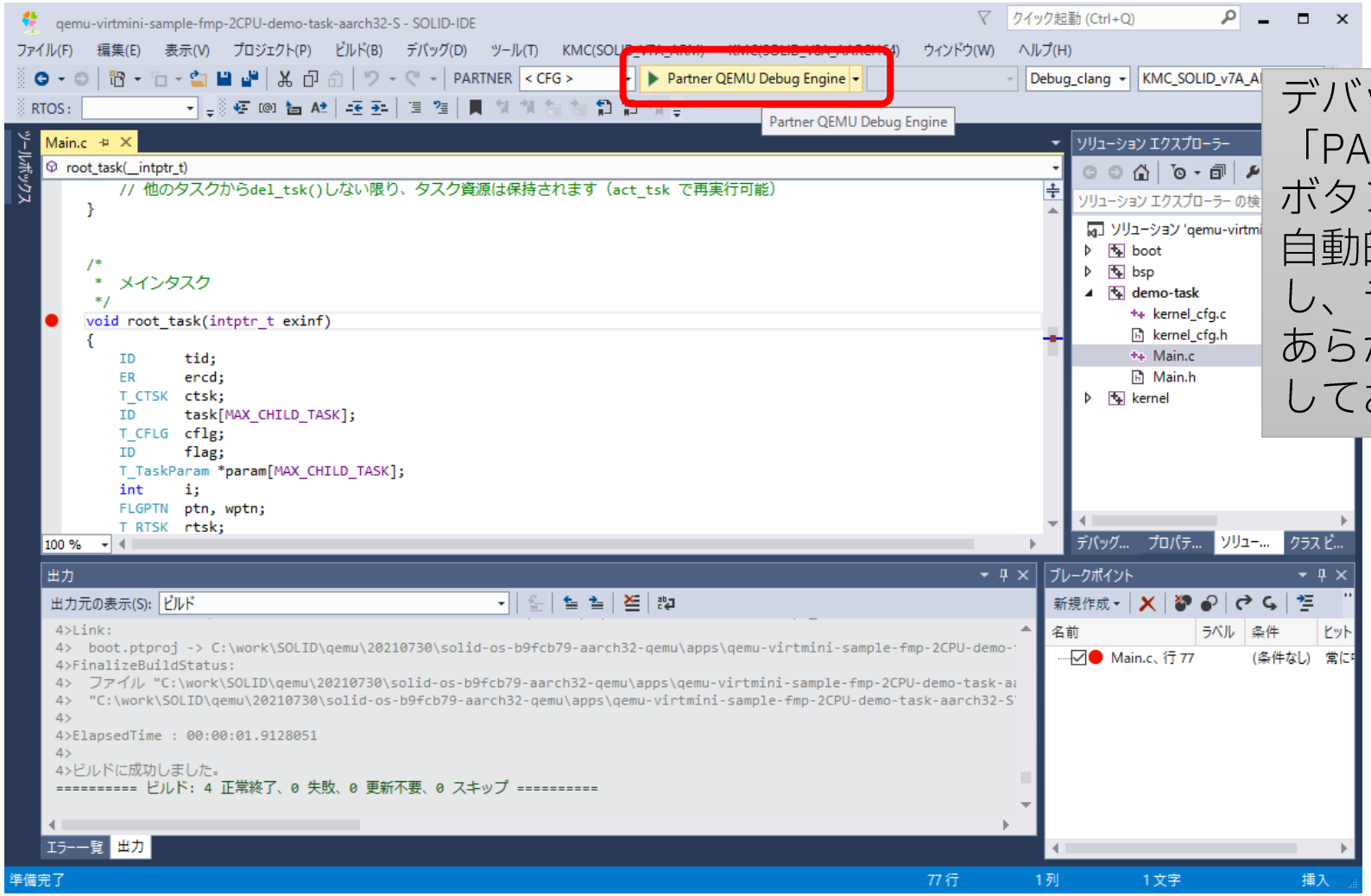
- “qemu-virtmini-sample-asp3-demo-task-*-S” を起動します
  - マルチコアカーネルで試したいときは、”qemu-virtmini-sample-fmp-demo-task-*-S“を起動してください
  - *は32bitを試したいときは aarch32を、64bitを試したいときは aarch64を選びます

ソリューションをビルドする



ソリューションのビルド、を実行すると、IDE下部にある出力ウィンドウに状況を表示しながらビルドを実行します

デバッグ実行を開始する

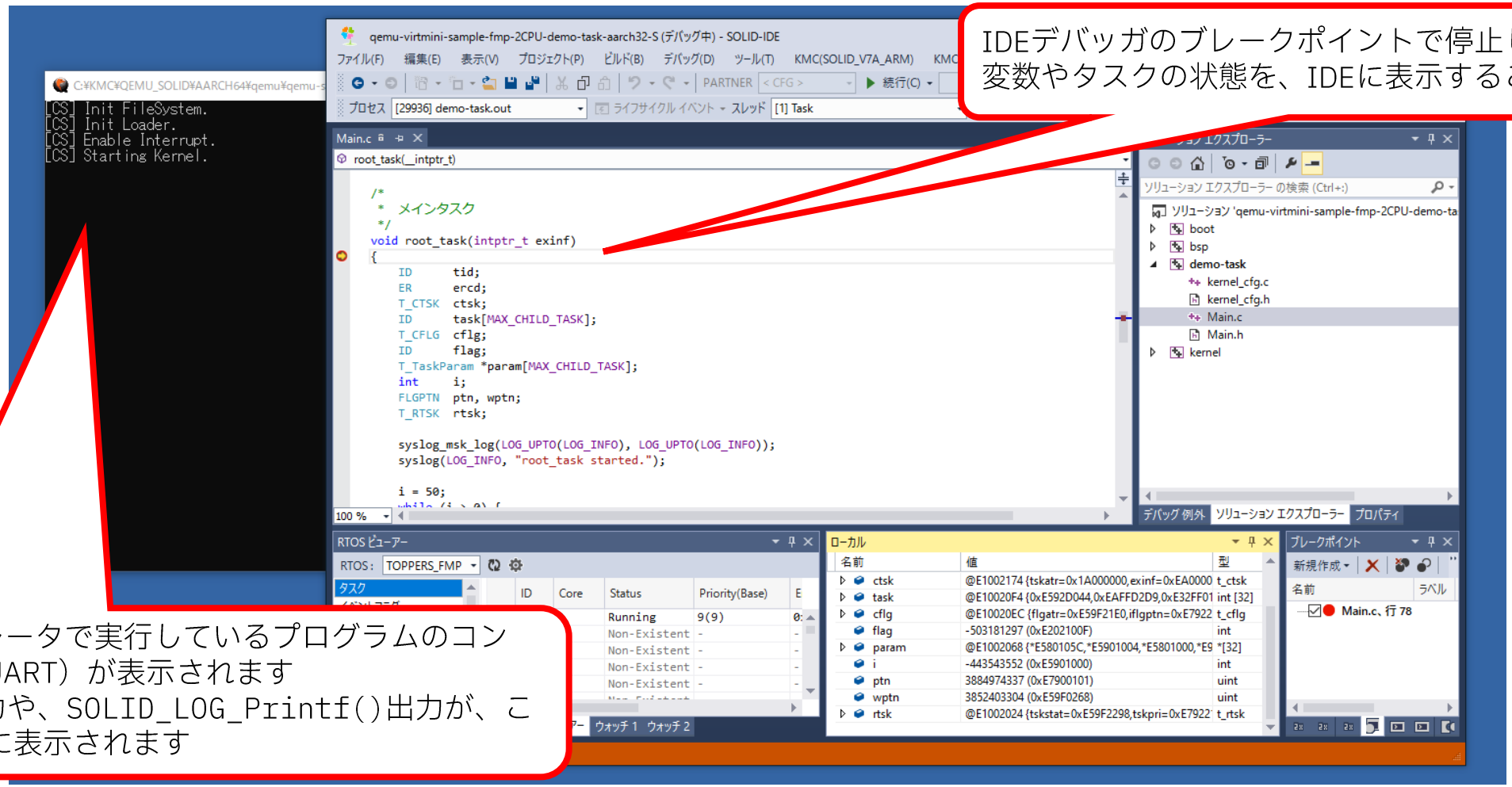


デバッグ実行を行うには、
「PARTNER QEMU Debug Engine」
ボタンを押下します。
自動的にQEMUシミュレータが動作し、デバッグ実行を開始します。
あらかじめブレークポイントを設定しておけば、そこで停止します

サンプルの動作について

- 初期タスク（root_task）が、子タスク（child_task）32個を動的に生成し、開始します
  - この時、マルチコア用サンプルは、CPU0/CPU1で動作するタスクを交互に生成し、開始します
- 子タスクは自分の処理が完了したら、イベントフラグを設定します
- 初期タスクは子タスクからのイベントフラグを待っており、全てのタスクが完了すると、全ての子タスクを削除します
- このあと、また子タスク32個の動的生成に戻り、永遠にこの処理を繰り返します

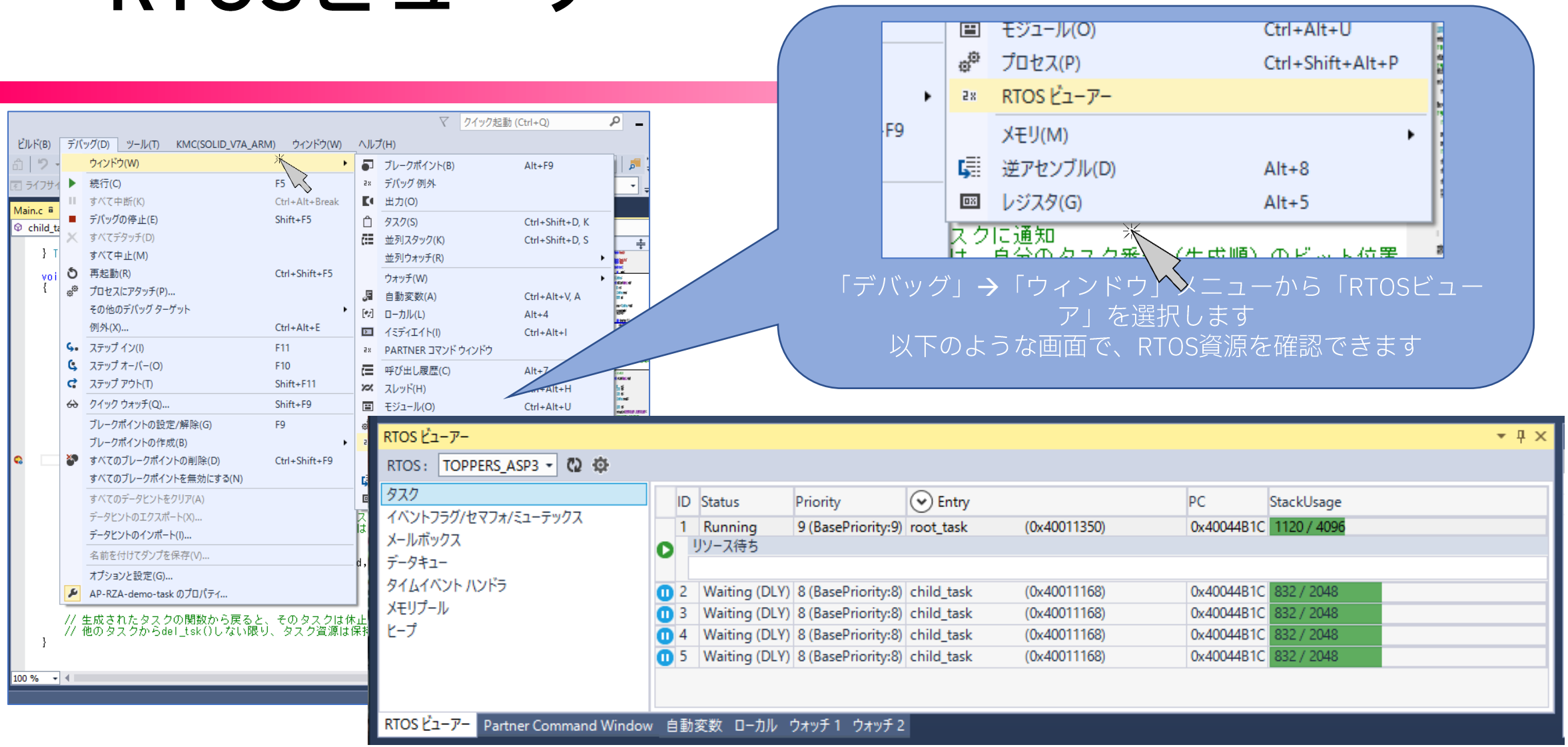
デバッグ実行の様子



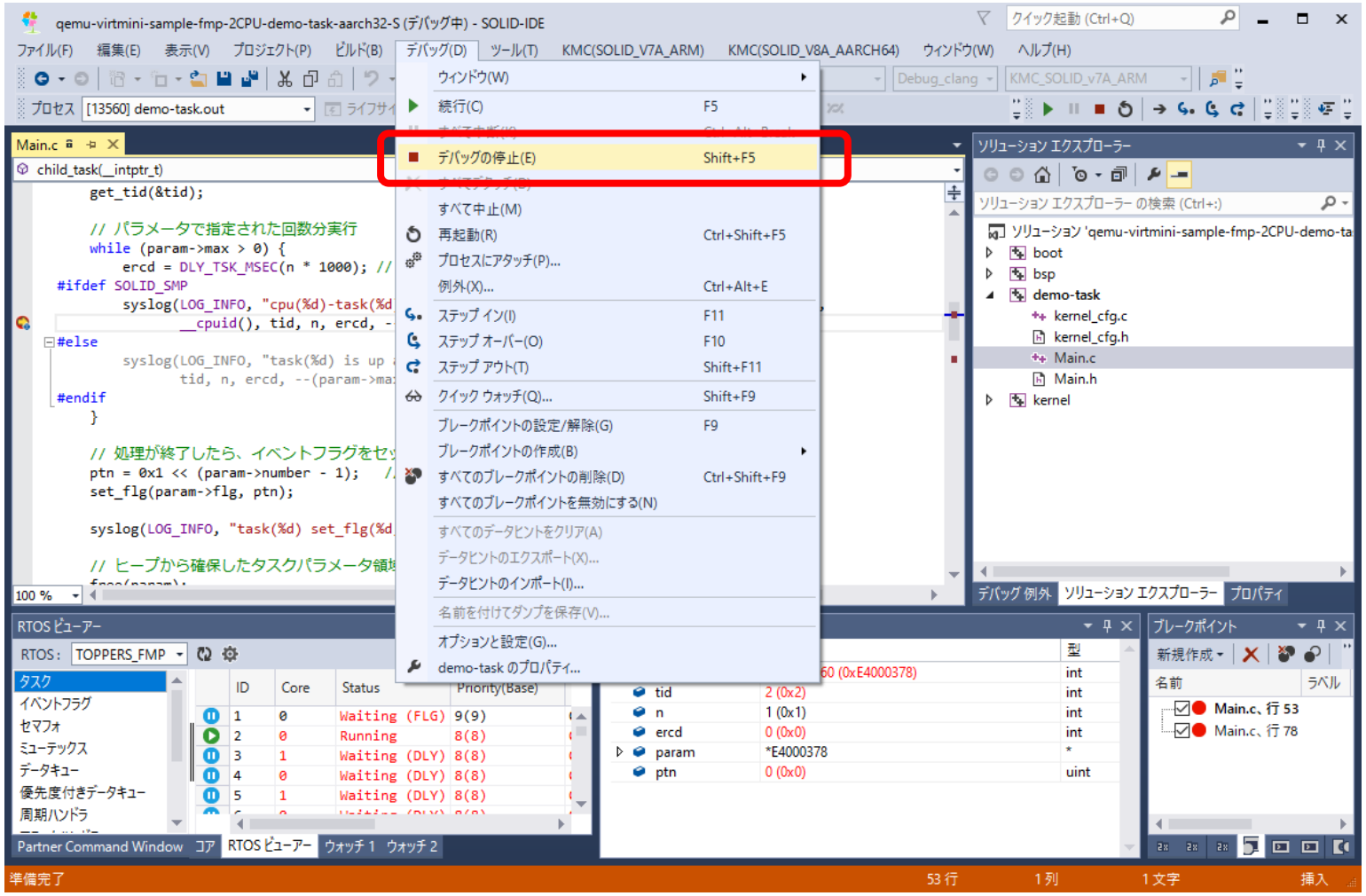
IDEデバッガのブレークポイントで停止しています。
変数やタスクの状態を、IDEに表示することができます

QEMUシミュレータで実行しているプログラムのコンソールログ（UART）が表示されます
syslog()出力や、SOLID_LOG_Printf()出力が、このコンソールに表示されます

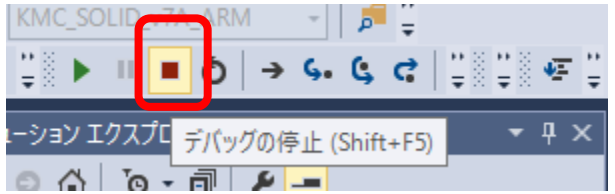
RTOSビューア


モジュール(O) Ctrl+Alt+U
プロセス(P) Ctrl+Shift+Alt+P
RTOS ビューアー
メモリ(M)
逆アセンブル(D) Alt+8
レジスタ(G) Alt+5
F9
「デバッグ」→「ウィンドウ」メニューから「RTOSビューア」を選択します
以下のような画面で、RTOS資源を確認できます
RTOS ビューアー
RTOS: TOPPERS_ASP3
タスク
イベントフラグ/セマフォ/ミューテックス
メールボックス
データキュー
タイムイベント ハンドラ
メモリプール
ヒープ
ID Status Priority Entry PC StackUsage
1 Running 9 (BasePriority:9) root_task (0x40011350) 0x40044B1C 1120 / 4096
リソース待ち
2 Waiting (DLY) 8 (BasePriority:8) child_task (0x40011168) 0x40044B1C 832 / 2048
3 Waiting (DLY) 8 (BasePriority:8) child_task (0x40011168) 0x40044B1C 832 / 2048
4 Waiting (DLY) 8 (BasePriority:8) child_task (0x40011168) 0x40044B1C 832 / 2048
5 Waiting (DLY) 8 (BasePriority:8) child_task (0x40011168) 0x40044B1C 832 / 2048
RTOS ビューアー
Partner Command Window
自動変数
ローカル
ウォッチ 1
ウォッチ 2

デバッグ実行の終了



「デバッグ」→「デバッグの停止」メニューを選択するか、もしくは、ツールバーの「デバッグの停止」ボタンを押下してください。



SOLIDインテリジェントローダー

ローダブルアプリ実行サンプル

- SOLIDのインテリジェントローダーを使い、別ロードモジュールとしてビルドしたローダブルアプリをロードして実行するサンプルです。
- インテリジェントローダーの概要をご理解いただく為に、まずは以下のURLの資料を一読して下さい
  - https://solid.kmckk.com/SOLID/archives/2166
- また、具体的な使い方は、以下に記載があります
  - http://solid.kmckk.com/doc/skit/current/tutorial/create_loadable_app_project.html

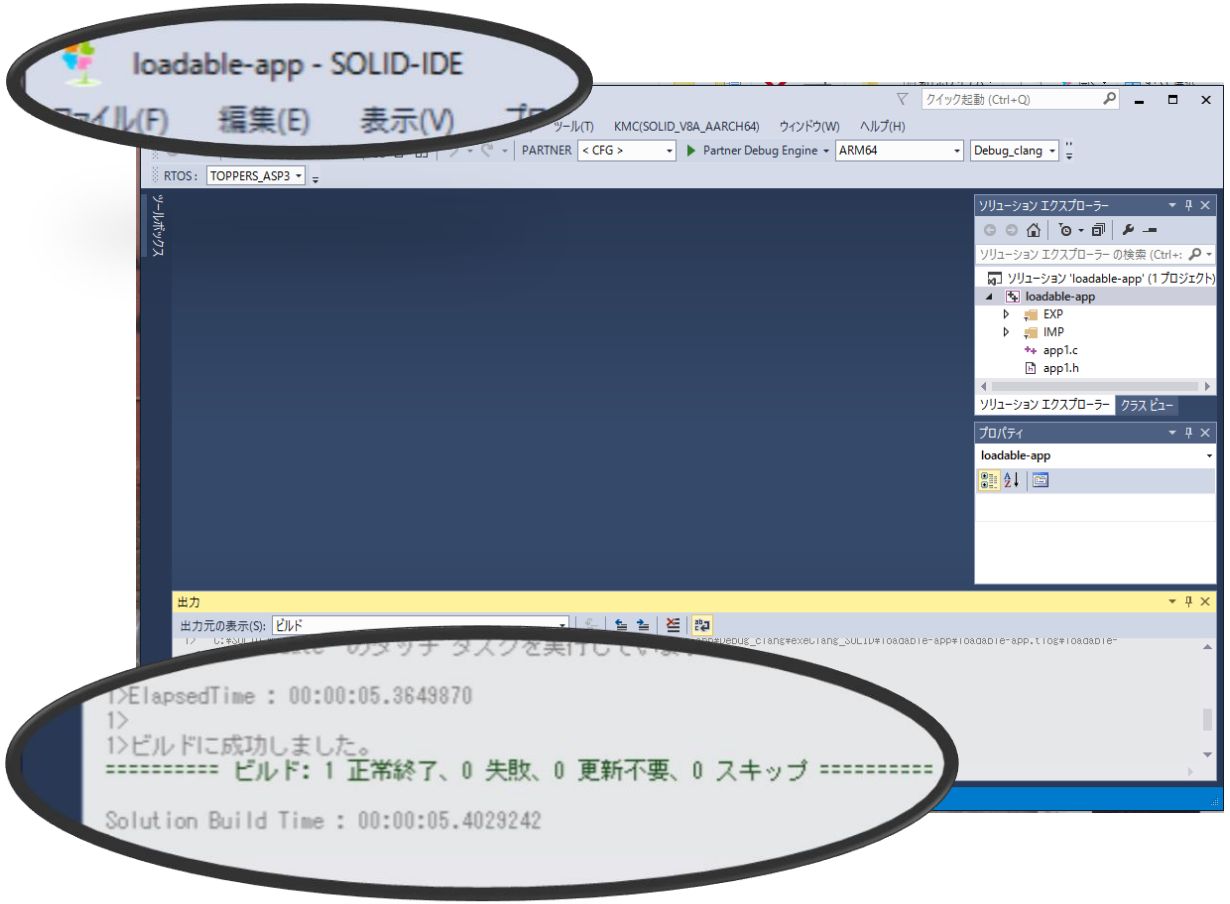
- SOLID独自形式のローダブルサンプルと、DLL形式のローダブルサンプル、の二種類を用意しています
  - 二種類の違いは、以下をご確認ください
    - http://solid.kmckk.com/doc/skit/current/tutorial/create_loadable_app_project.html#id4

SOLID独自形式ローダブルサンプル

ローダブルアプリ実行サンプル

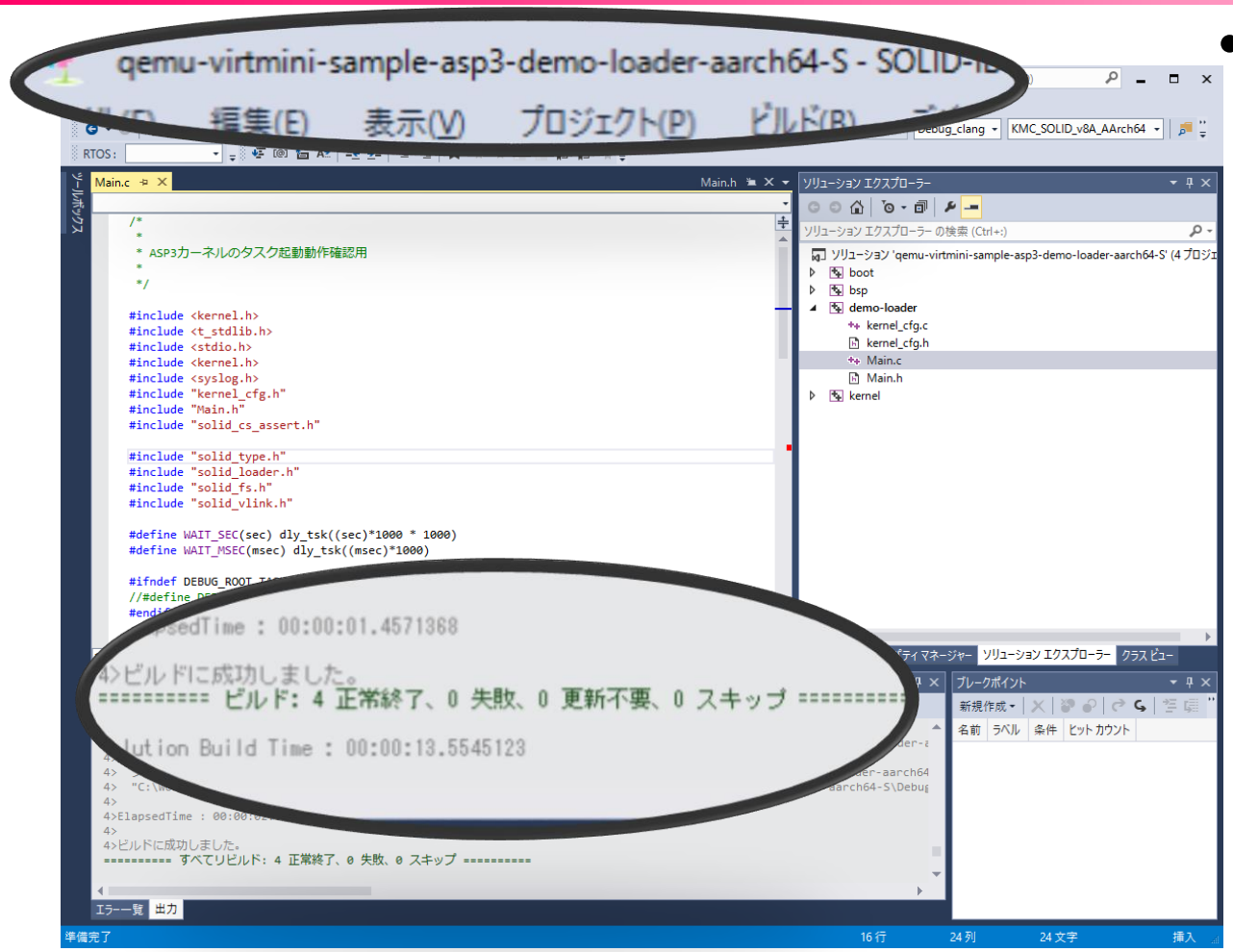
- 以下のサンプルを使います
  - `qemu-virtmini-sample-asp3-demo-loader-*-S`
- サンプルの実行は以下の手順で行います
  1. ローダブルアプリをSOLID-IDEでビルドします
     - `qemu-virtmini-sample-asp3-demo-loader-*-S¥loadable-app¥loadable-app.ptsln`
  2. 上記アプリをロードするメインアプリ側をSOLID-IDEでビルドします
     - `qemu-virtmini-sample-asp3-demo-loader-*-S.ptsln`
  3. メインアプリのSOLID-IDEでデバッグを実行します
     メインアプリの内部で、ローダブルアプリをロードしてメインアプリの一部として実行します。
- その後
  1. ローダブルアプリを一部変更してビルド
  2. メインアプリを再ビルドすることなく同じ SOLID-IDEで実行することで上記変更が反映されることが確認できます

ローダブルアプリのビルド



- 以下のソリューションを起動し、ビルドします
  - `qemu-virtmini-sample-asp3-demo-loader-*-S¥loadble-app¥loadable-app.ptsln`

メインアプリのビルド



- 以下のソリューションを起動し、ビルドします
  - `qemu-virtmini-sample-asp3-demo-loader-*-S.ptsln`

メインアプリの実行

以下のような箇所にブレークポイントを設定し、
SOLIDローダーの動きを確認することができます。

今回、ローダブルアプリをロードするためのファイルシステムは、デバッガと連携してWindows上のファイルを直接にアクセスできる、VLINKというファイルシステムを使っています。
ここで、Windows上でビルドしたディレクトリのパスを取得し、ロードしたいローダブルアプリのファイルパスを生成しています。

Main.c

root_task(__intptr_t)

```
{
    syslog_msk_log(LOG_UPTO(LOG_INFO), LOG_UPTO(LOG_INFO));
    syslog(LOG_INFO, "root_task() is up.\n");

    if (SOLID_VLINK_GetCwd(vlinkbuf) == SOLID_ERR_OK) {
        *strrchr(vlinkbuf, '\\') = '\0';
        sprintf(filename, "\\VLINK\\%s\\%s", vlinkbuf, LOAD_APP_NAME);
    }

    int ret = SOLID_LDR_LoadFile("APP1", filename);
    if (ret == SOLID_ERR_OK) {
        SOLID_ADDRESS addr;
        void (*dispfunc)(void);
        ret = SOLID_LDR_CanExec("APP1", &addr);
        if (ret > 0) {
            if (SOLID_ERR_OK == SOLID_LDR_GetAddr("appFuncAddrDisp", (SOLID_ADDRESS *)(&dispfunc))) {
                dispfunc();
            }
            int (*pFunc)(void) = (int (*)(void))addr;
            ret = pFunc();
        }
```

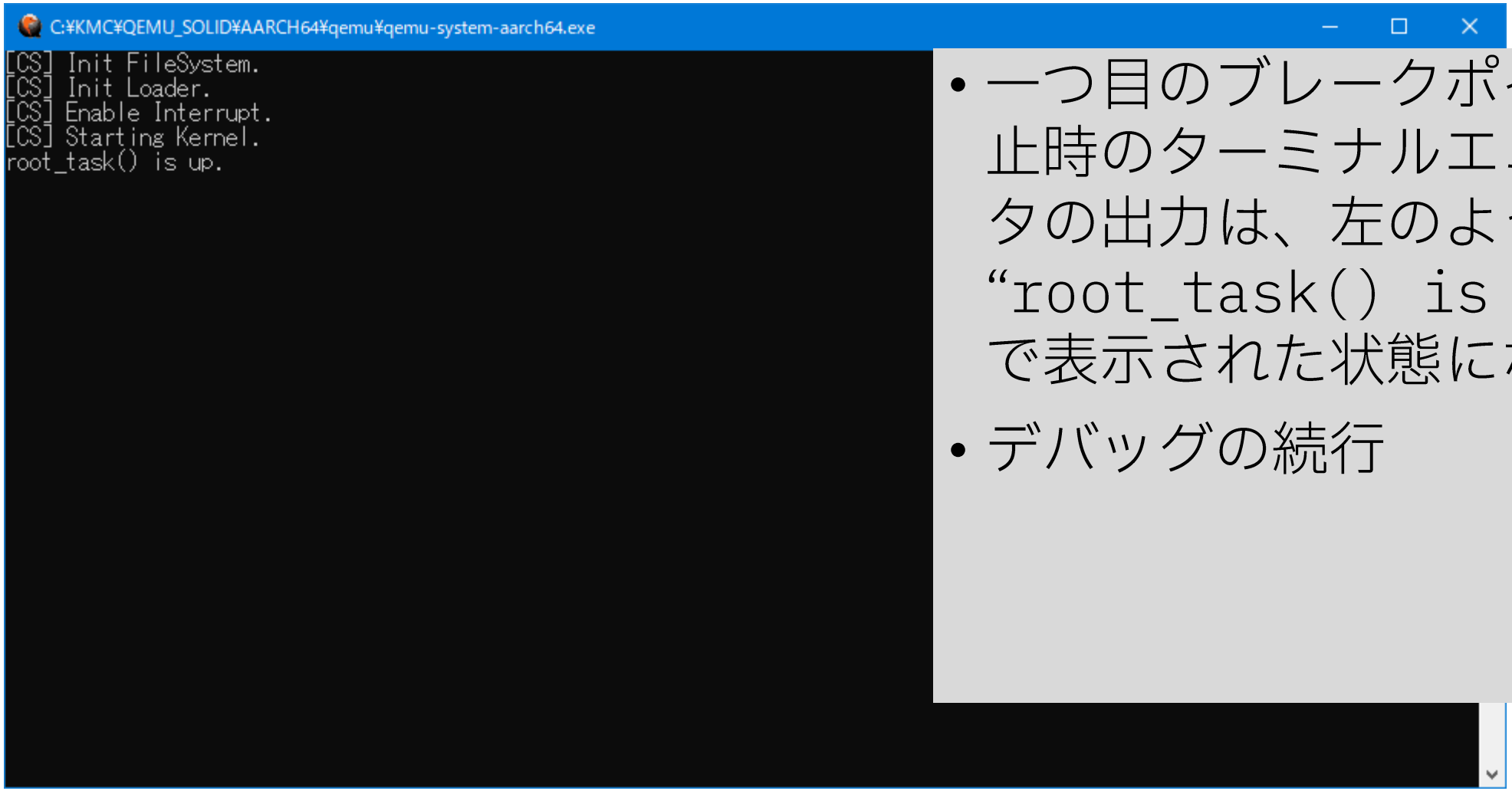
100 %

指定のパスのファイルを、ローダブルアプリとしてロードします。

ローダブルアプリの公開関数のアドレスを取得し、その関数を呼び出してみます

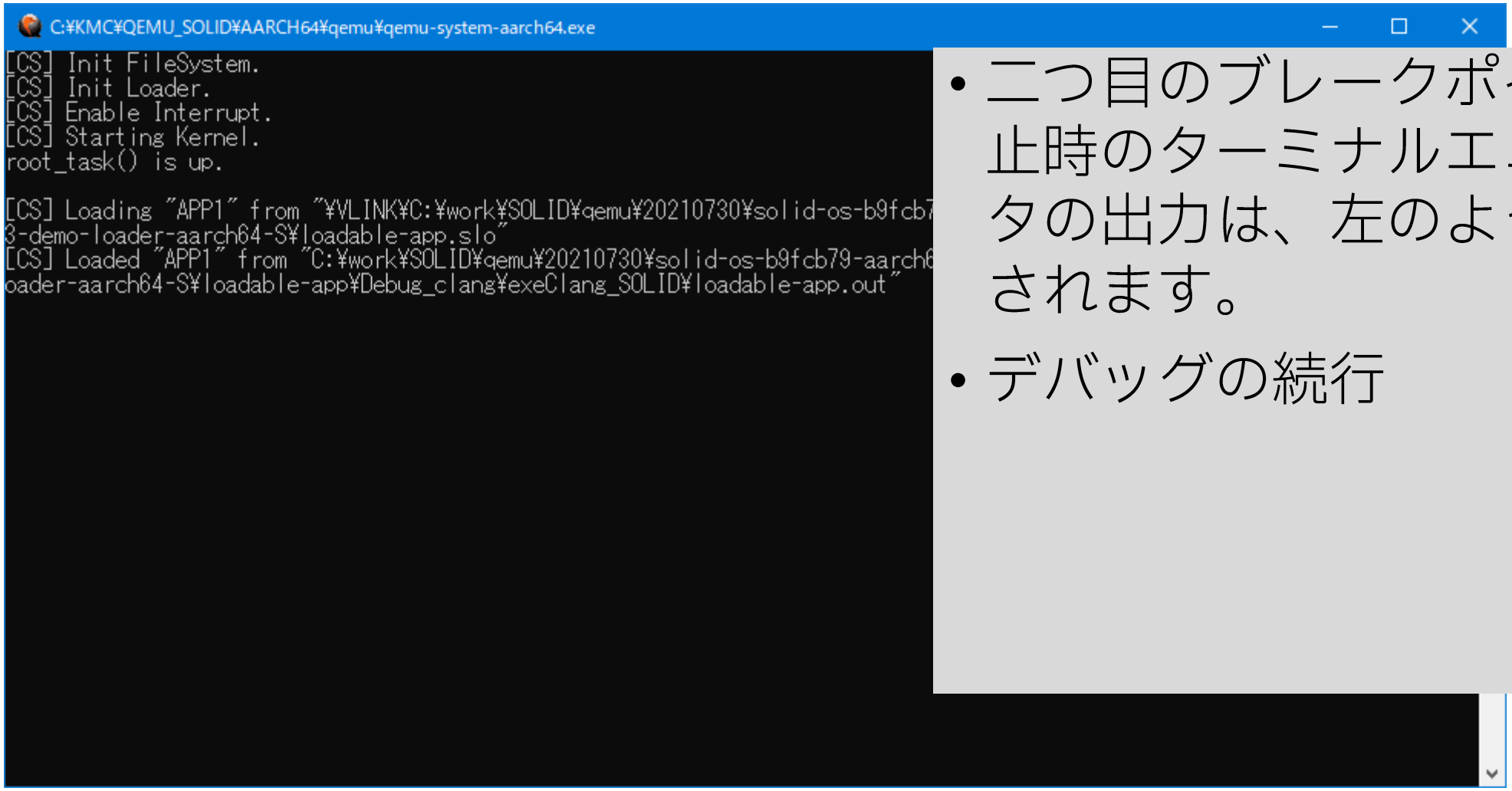
ローダブルアプリのエントリアドレス（CanExec APIが成功したときに取得したアドレス）を使って、ローダブルアプリを実行してみます

一つ目のブレークポイント停止時



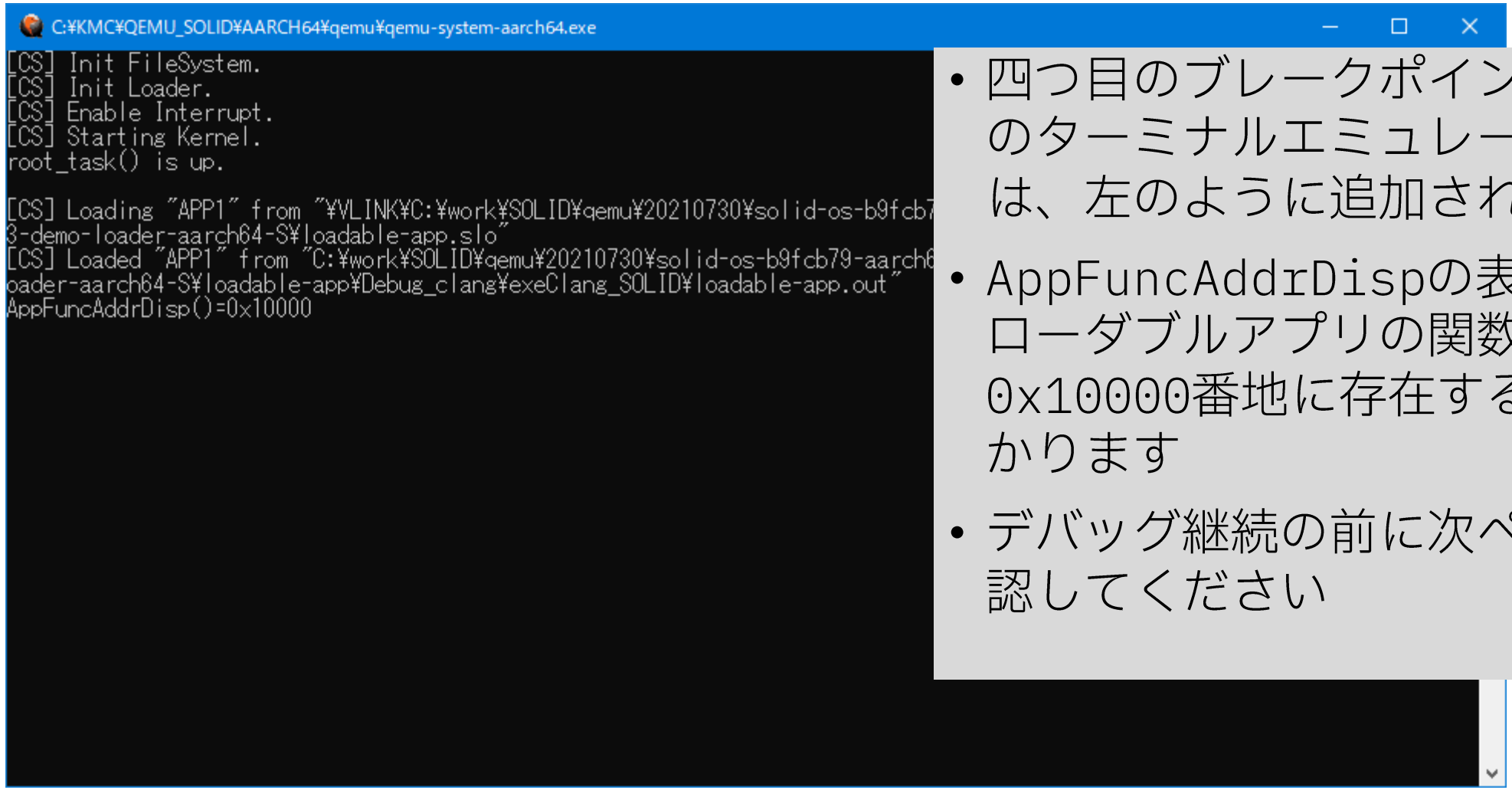
- 一つ目のブレークポイント停止時のターミナルエミュレータの出力は、左のような、“root_task() is up.”まで表示された状態になります。
- デバッグの続行

二つ目のブレークポイント停止時



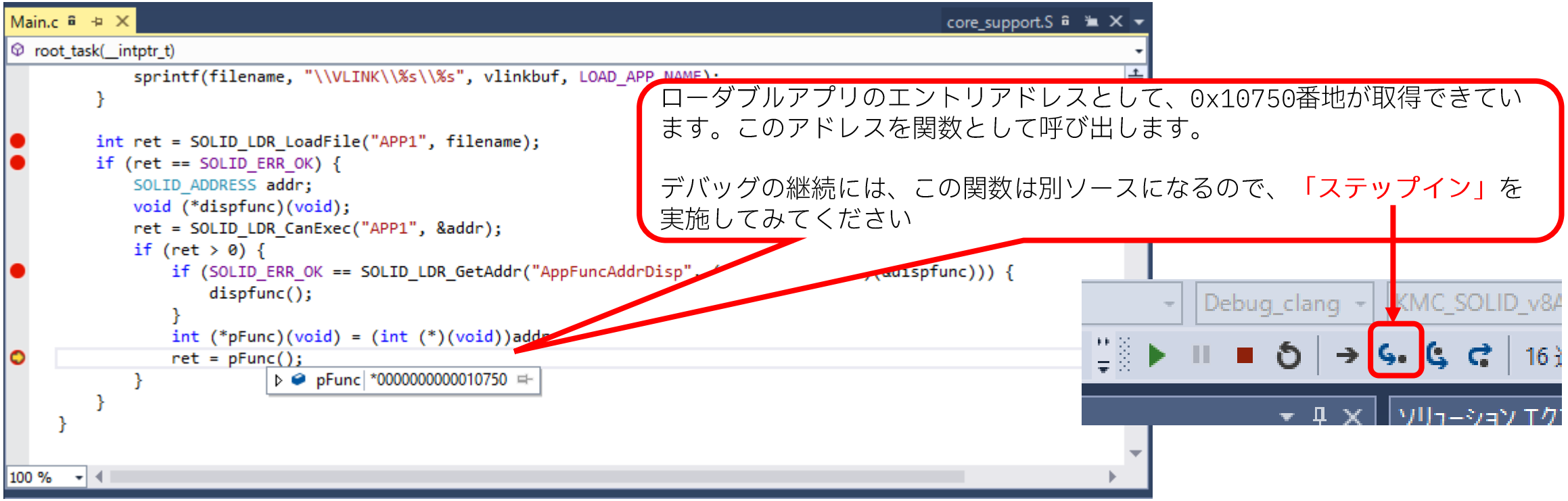
- 二つ目のブレークポイント停止時のターミナルエミュレータの出力は、左のように追加されます。
- デバッグの続行

四つ目のブレークポイント停止時

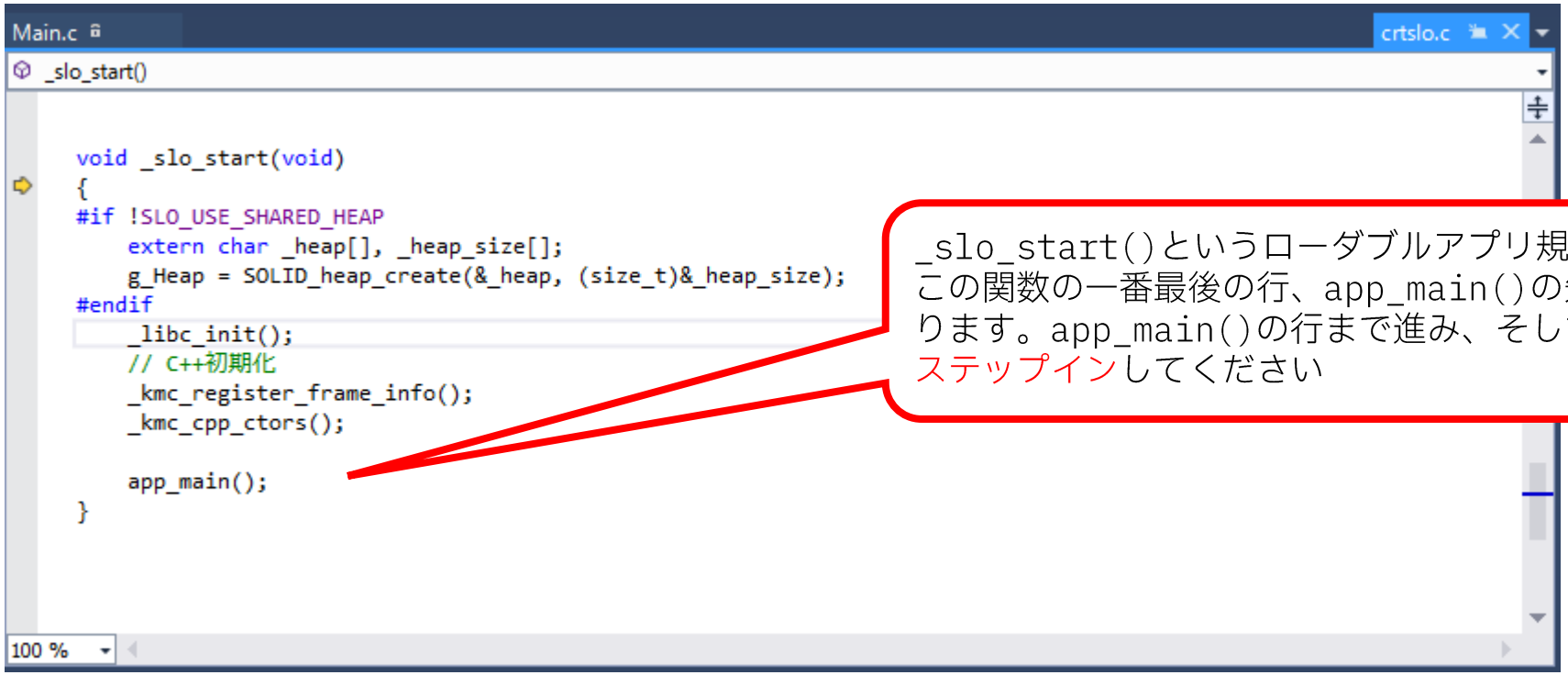


- 四つ目のブレークポイント停止時のターミナルエミュレータの出力は、左のように追加されます。
- AppFuncAddrDispの表示から、ローダブルアプリの関数は0x10000番地に存在することがわかります
- デバッグ継続の前に次ページを確認してください

四つ目のブレークポイント停止時



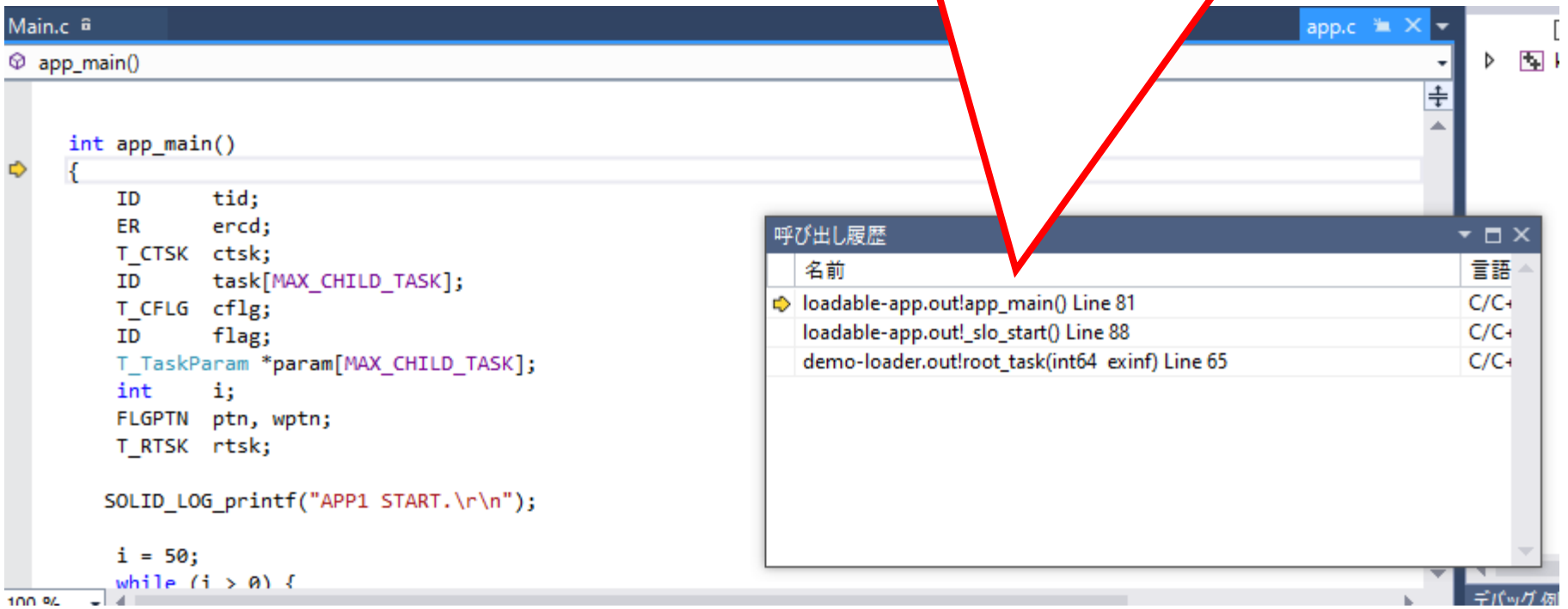
ローダブルアプリにステップインすると、、、



_slo_start()というローダブルアプリ規定の初期化処理が呼び出されます。この関数の一番最後の行、app_main()の先が、ユーザーコードの先頭になります。app_main()の行まで進み、そして、もういちど、app_main()にステップインしてください

app_main()にステップインすると、、、

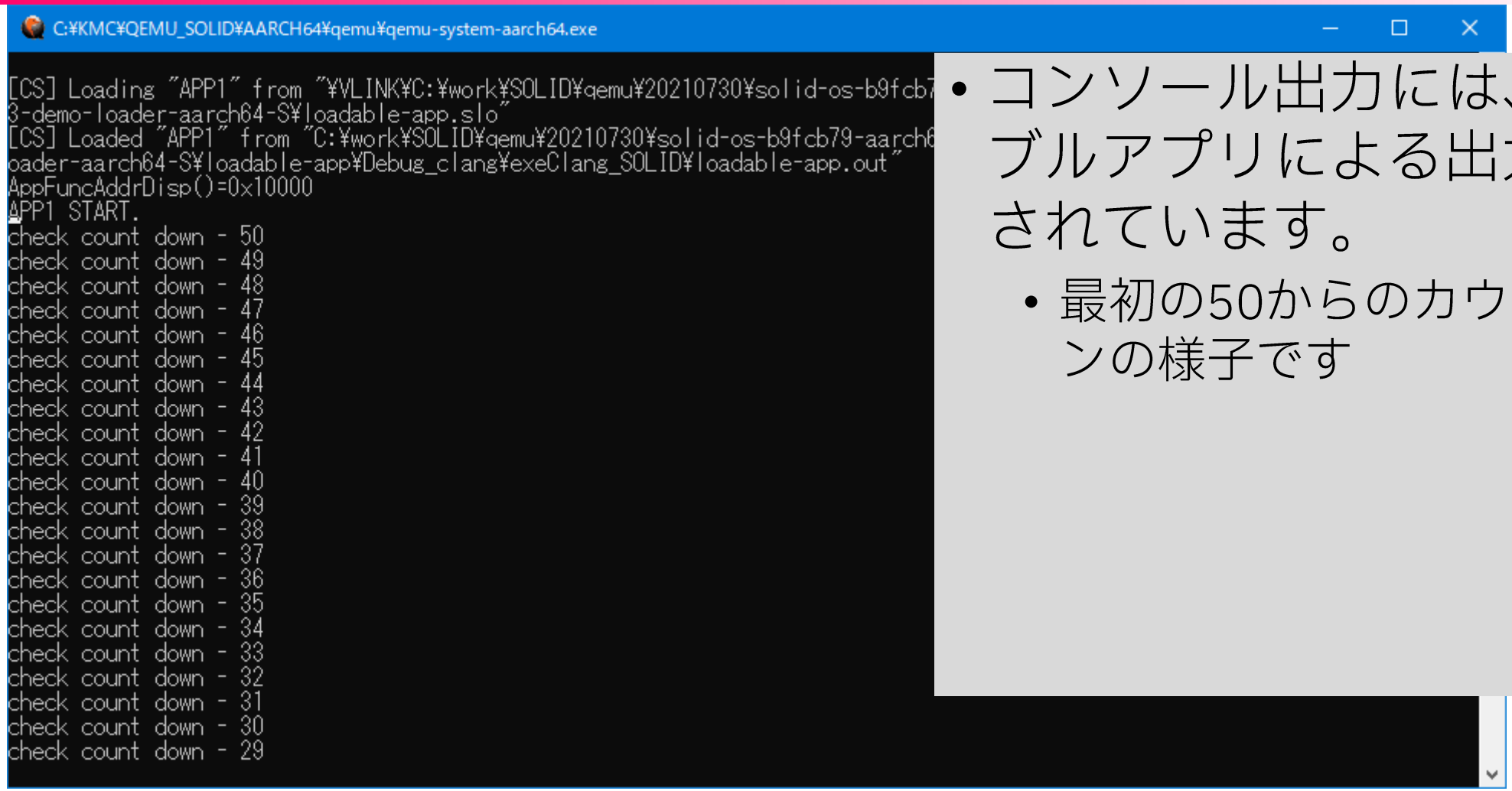
ローダブルアプリのユーザーコードの先頭にきました。
ここで、「デバッグ」→「ウィンドウ」→「呼び出し履歴」を表示すると、メインアプリのroot_task()から、ローダブルアプリのapp_main()まで関数呼び出し履歴が繋がっていることが分かります



ローダブルアプリのデバッグと実行

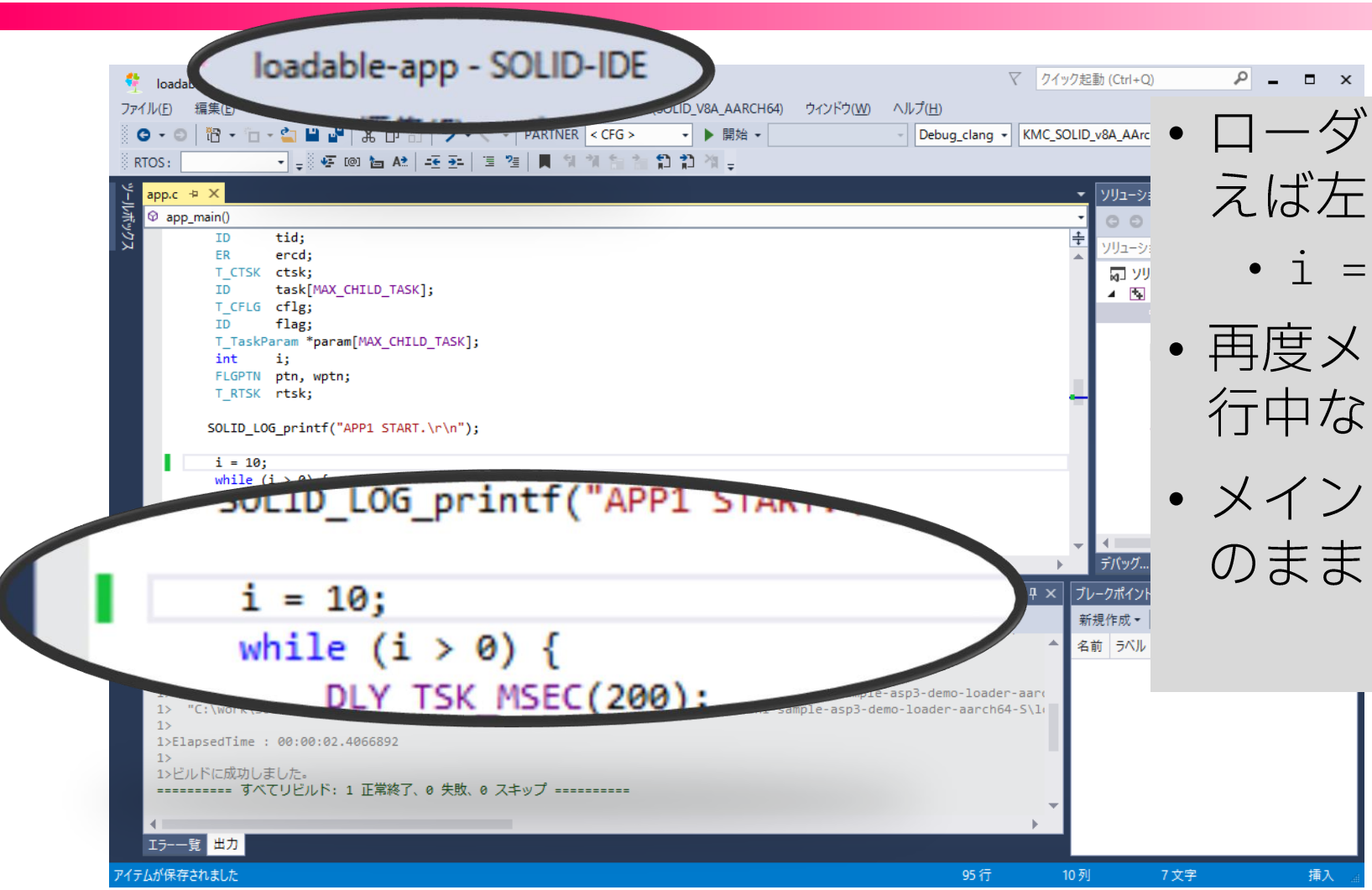
- ローダブルアプリの内容は、「タスク切り替えサンプル」と同じ内容のプログラムが動作します
- タスク切り替えサンプルからは、RTOSのAPI（acre_tsk、sta_tsk、set_flg、....）が呼び出されていますが、これらのAPIの呼び出し先はメインアプリ側に存在します
  - もちろん、実行について、何の制約もなく実行することができます
  - また、これら、メインアプリ側の存在するAPIをまたがっても、通常にステップインなどのデバッグすることが可能です

ローダブルアプリを実行継続



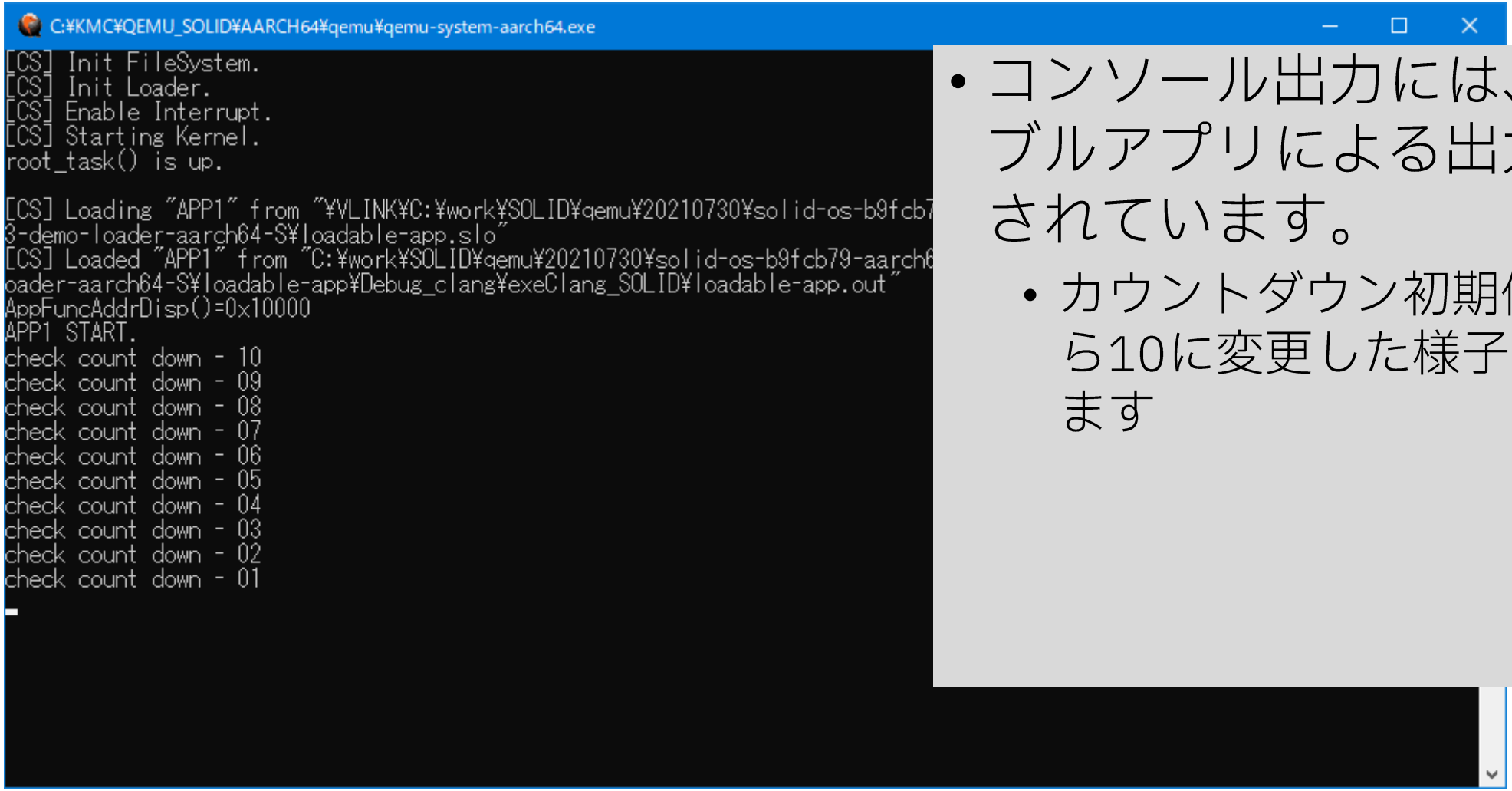
- コンソール出力には、ローダブルアプリによる出力が表示されています。
  - 最初の50からのカウントダウンの様子です

ローダブルアプリのみの書き換え



- ローダブルアプリのIDEを開き、例えば左のように書き換えてビルド
  - `i = 50`を`i = 10`に変更しています
- 再度メインアプリのIDEを開き（実行中ならデバッグ停止をします）
- メインアプリは再ビルドせず。そのままデバッグの実行をします

ローダブルアプリを実行継続



- コンソール出力には、ローダブルアプリによる出力が表示されています。
  - カウントダウン初期値を50から10に変更した様子がわかります

DLL形式ローダブルサンプル

DLLアプリ実行サンプル

- 以下のサンプルを使います
  - qemu-virtmini-sample-asp3-demo-dll-*-S
- サンプルの実行は以下の手順で行います
  1. DLLアプリをSOLID-IDEでビルドします
     - qemu-virtmini-sample-asp3-demo-dll-*-S¥dll_app1¥dll_app1.ptsln
  2. 上記アプリをロードするメインアプリ側をSOLID-IDEでビルドします
     - qemu-virtmini-sample-asp3-demo-dll-*-S.ptsln
  3. メインアプリのSOLID-IDEでデバッグを実行します
     - メインアプリの内部で、DLLアプリをロードして実行します
     - この時、DLLの特徴である、同じDLLを二重にローディングして、異なるアドレスで複数実行できることを確認します

DLLサンプルのビルドと実行

- ビルドについては、SOLID独自形式のサンプル、と同じような手順でビルドをしてください
  - DLL側をビルド
  - メインアプリをビルド
- デバッグはメインアプリ側のIDEから行います
- 次ページのようにブレークポイントを設定して、試してみてください

DLLアプリのローディングを確認

Main.c

root_task(__intptr_t)

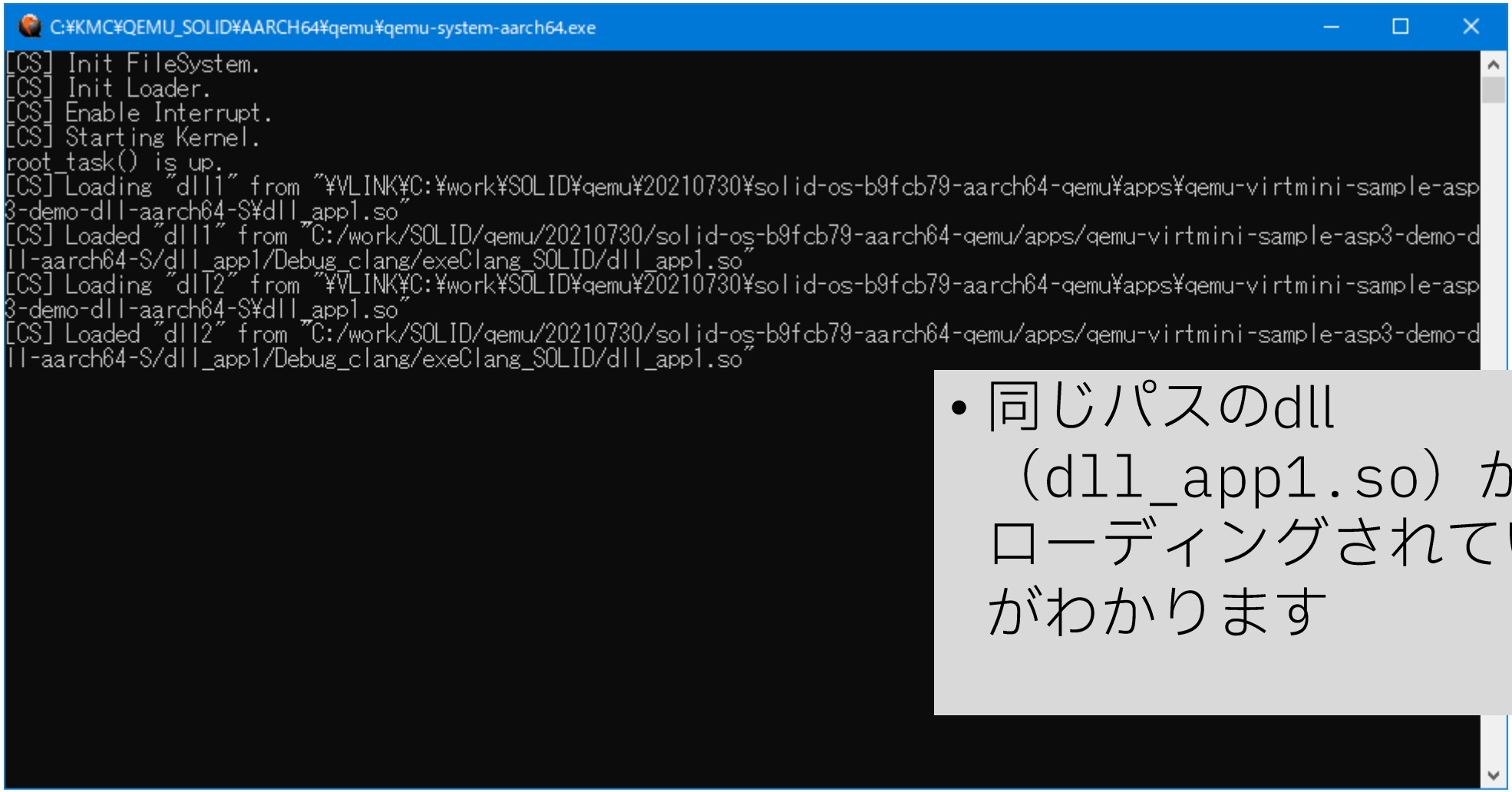
```
        ret = SOLID_LDR_LoadDLL("dll1", filename, SOLID_LDR_ADDR_DLLAREA );
        if (SOLID_ERR_OK != ret) {
            SOLID_LOG_printf("SOLID_LDR_LoadDLL(%s), error=%d\n", filename, ret);
            return;
        }
        ret = SOLID_LDR_LoadDLL("dll2", filename, SOLID_LDR_ADDR_DLLAREA );
        if (SOLID_ERR_OK != ret) {
            SOLID_LOG_printf("SOLID_LDR_LoadDLL(%s), error=%d\n", filename, ret);
            return;
        }

        if (SOLID_ERR_OK == SOLID_LDR_GetDllAddr("dll1", "AppFuncAddrDisp", (SOLID_ADDRESS *)(&dispfunc))) {
            dispfunc();
        } else {
            SOLID_LOG_printf("SOLID_LDR_GetDllAddr(%s, %s), error=%d\n", "dll1", "AppFuncAddrDisp", ret);
        }
        if (SOLID_ERR_OK == SOLID_LDR_GetDllAddr("dll2", "AppFuncAddrDisp", (SOLID_ADDRESS *)(&dispfunc))) {
            dispfunc();
        } else {
            SOLID_LOG_printf("SOLID_LDR_GetDllAddr(%s, %s), error=%d\n", "dll2", "AppFuncAddrDisp", ret);
        }

    retry:
        ret = SOLID_LDR_CanExec("dll1", &addr);
        if (ret > 0) {
```

100 %

指定のパスのファイルを、DLLとしてロードします。
アプリ名を変更して（dll1とdll2）、二回ローディングします

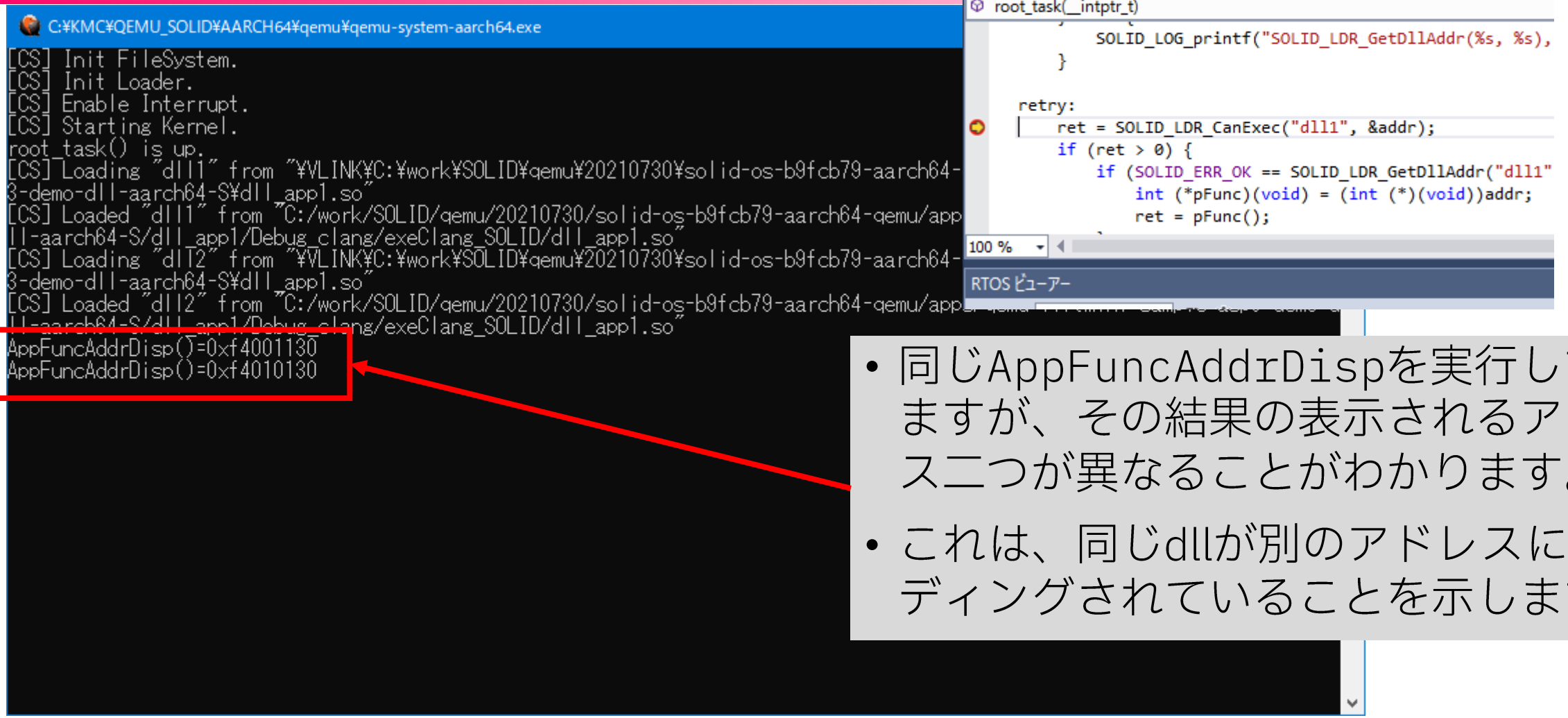
ローディングしたDLLの公開シンボル”AppFuncAddrDisp”を検索します。
アプリ名を変更して（dll1とdll2）二回検索します

二つ目のブレークポイントまで実行



- 同じパスのdll（dll_app1.so）が二回ローディングされていることがわかります

五つ目のブレークポイントまで実行



- 同じAppFuncAddrDispを実行していますが、その結果の表示されるアドレス二つが異なることがわかります。
- これは、同じdllが別のアドレスにローディングされていることを示します

おわり