文書番号: PDTSC20131106-029-003

SIOS Protection Suite for Linux GenericARK 開発ガイド

～LifeKeeper for Linux v8 以降対応版～

第 1.2 版

目次

改訂履歴

2013 年 11 月 6 日	初版
2016 年 9 月 6 日	第 1.1 版：LifeKeeper v9 もサポート
2017 年 6 月 5 日	第 1.2 版：LifeKeeper v9.1.0 GenericARK タイムアウト対応

1. はじめに

LifeKeeper for Linux では、Application Recovery Kit(ARK)が対応していないアプリケーションを保護する仕組みとして、GenericARK を用意しています。GenericARK は、ユーザー自身の手で制御スクリプトを作成し、GenericARK リソースとして LifeKeeper に組み込み、LifeKeeper の保護対象とします。

本書では、実際のスクリプトの中で検討するべき事項や、LifeKeeper の特性、連携についてサンプルコードを交えて案内いたします。

本文書の内容、対象となる読者は以下の前提に従います。

- 本文書の内容は、LifeKeeper for Linux テクニカルトレーニングを受講したか、または同等程度の知識をお持ちであることを前提に書かれています。
- 本文書において、GenericARK リソースとして保護する対象は、一般的なサーバアプリケーション(コマンドラインから起動/停止の制御が可能であり、クライアントに対して何らかのサービスを提供するもの)を想定しています。
- 記載内容は LifeKeeper for Linux v8 以降を対象とします。
- サンプルコードは bash によるシェル・スクリプトを使用します。

2. 表記上の約束について

本書では数種の表記上の約束を採用しています。

- typewriter はターミナルやコンソール上画面に表示されるシステムからのメッセージまたはプロンプトを表します。

```
Configuration successfully completed.
```

- ターミナルやコンソール上画面に使用されるプロンプトは以下の意味を表します。
  例:コマンドがスーパーユーザーで実行される場合

```
#
```

  例:コマンドが一般ユーザーで bash または ksh で実行される場合

```
$
```

- 文章中の【】に囲まれている単語は、ユーザーが入力するキーおよび GUI 上での操作を表します。
  例:【Return】キーを入力してください。

3. GenericARK の仕様

GenericARK のスクリプトに記述する内容について、LifeKeeper は特に制限を設けておりません。しかし、保護対象のアプリケーションをユーザーの期待通りに制御するためには、GenericARK の仕組みや LifeKeeper の仕様についての理解が必要です。

ここでは、GenericARK リソースを作成する際に必要となる LifeKeeper と GenericARK の仕様について紹介します。

3-1. 各スクリプトのタイムアウト

GenericARK に登録したスクリプトは、ハングアップに備えてタイムアウトの機能を有すべきです。v9.1.0 以降の LifeKeeper は、各スクリプトに対してタイムアウトの仕組みを提供します。v9.0.2 までの LifeKeeper は、quickCheck に対してのみタイムアウトの仕組みを提供します。
v9.0.2 までの LifeKeeper で quickCheck 以外のスクリプトに対して、またタイムアウト時に独自の処理を実行したい場合は、スクリプト内にタイムアウトの仕組みを実装してください。以下にタイムアウトのサンプルスクリプトを記載致します。

サンプルコード 1：タイムアウトの実装

```
#!/bin/bash

DEFAULT_TIMEOUT=22    # スクリプトタイムアウト値


# タイムアウト設定
TTAG=$(echo $TAG | sed "s/[^A-Za-z0-9_]/_/g")
TTAG_TIMEOUT=$(eval echo '$'$TTAG'_TIMEOUT')
if [ -z "$TTAG_TIMEOUT" ]; then
        TIMEOUT=$DEFAULT_TIMEOUT
else
        TIMEOUT=$TTAG_TIMEOUT
        log LK_DEBUG 1 "Timeout set to $TIMEOUT sec."
fi

# タイムアウト適用処理
# 引数: スクリプトのタイムアウト秒数
setalarm()
{
        (
                sleep $1;
                (ps $$ | grep $0) && kill -INT $$;
                (ps $$ | grep $0) && sleep $1;
                (ps $$ | grep $0) && kill -KILL $$
        ) > /dev/null 2>&1 &


# タイムアウト発生時後処理
handlealarm()
{
        log LK_ERROR 5 "Script was hung. Forcibly terminating."

        # 停止処理実施
        APP_stop
        APP_force_stop

        # 自身の停止処理 (SIGTERM -> SIGKILL)
        if [ "$(ps $$ > /dev/null 2>&1; echo $?)" -ne 1 ]
        then
                kill -TERM $$
                sleep 5
                if [ "$(ps $$ > /dev/null 2>&1; echo $?)" -ne 1 ]
                then
                      log LK_ERROR 6 "Script was still hung. sending KILL signal"
                      kill -KILL $$
                fi
        fi
        exit 1
}
```


3-2. エラー時のリトライ

GenericARK には、restore, remove, quickCheck, recover の各スクリプトの実行において、自動的にリトライを行うことはありません。処理の結果に応じてリトライを行う必要がある場合は、スクリプトの内部にリトライの仕組みを作成する必要があります。
※サンプルコードが「4-3 サンプルコード 2」にあります。参考としてください。

3-3. フェイルオーバーのトリガーと切り替わりの動作

LifeKeeper において、フェイルオーバー(障害検知によるリソース切り替え)のトリガーとなる事象は、大きく分けて以下の 2 種類があります。

1. 各リソースの quickCheck により障害を検知し、待機ノードにフェイルオーバーを行う。
2. ハートビートの全断により相手ノードが障害を検知し、待機ノードにフェイルオーバーを行う。

項目 1 の「各リソースの quickCheck によるフェイルオーバーの発生」では、以下のようにリソースの制御が行われます。

1. quickCheck スクリプトにより、障害を検知する。
2. recover スクリプトにより、障害を検知したノード上でリソースの再起動を行う。
3. recover でサービスが回復しなかった場合は、recover の失敗となり、待機ノードへのフェイルオーバーの実行を決定する。
4. 障害を検知したノード上で、フェイルオーバーの対象となる全リソースの remove(停止)が行われる。
5. 上記 4 の停止処理が完了した後、フェイルオーバー先のノード上で、必要なリソースの restore(起動)が行われる。

また、項目 2 の「ハートビート切断によるフェイルオーバー」では、以下のような制御となります。

1. 待機状態のノードがハートビートの全断を検知し、待機ノードへのフェイルオーバーの実行を決定する。
2. フェイルオーバー先のノード上で、必要なリソースの起動を行う。

このように、フェイルオーバー時の動作は、リソース障害に起因したものか、ハートビート障害に起因したものかによって、切り替わり時の動作が変わります。特に、リソースの停止処理が含まれるかどうかは大きな違いです。

3-4. 仮想 IP アドレスと仮想ホスト名

通常、HA クラスタを構成した場合、サービスにアクセスするクライアントは、仮想 IP アドレスか、仮想 IP アドレスにバインドされた仮想ホスト名に対してアクセスを行います。

GenericARK で保護するアプリケーションにおいても、仮想 IP アドレスに紐づいた仮想ホスト名が必要となるケースがあります。そのような設定が必要となるかどうかは、事前にアプリケーション側の要件を確認しておくことをお勧めいたします。

仮想 IP アドレスに紐付く仮想ホスト名が必要である場合は、/etc/hosts や DNS など、通常の名前解決のシステムで対応してください。

4. 起動(restore)スクリプトの実装

restore スクリプトにおいて、一般的なアプリケーションを保護する際の留意点について説明します。restore スクリプトを作成する場合、以下の点について配慮が必要です。

❒起動対象のサービスが、既に起動状態である場合への配慮
❒起動処理および起動後の状態確認
❒最終的な戻り値の作成
❒リトライの制御
❒restore のタイムアウト

4-1. 起動対象のサービスが、既に起動状態である場合への配慮

LifeKeeper は起動の際に、最後に停止した時のステータスを再現するべく、必要に応じてリソースの起動を行います。OS の起動に伴う LifeKeeper の自動起動や、”lkstart”コマンドで LifeKeeper を単体で起動した場合でも同様の動作になります。

ここで注意することは”lkstop -f”コマンドで LifeKeeper を停止した場合です。”lkstop -f”コマンドは、稼動中のサービスは停止せずに、LifeKeeper のプロセスのみを停止させます。このコマンドは、主に保護しているアプリケーションのメンテナンスを行う際に利用します。

“lkstop -f”コマンドで LifeKeeper を停止した後、LifeKeeper のプロセスを稼働させる場合は、”lkstart”コマンドを実行します。”lkstart”コマンドを実行すると、LifeKeeper は各リソースの restore スクリプトを実行します。これは”lkstop -f”コマンドを実行する直前のステータスが「起動中(ISP)」であることが理由です。

製品として発売している各リカバリキットにおいては、”lkstop -f”コマンド後に "lkstart”コマンドを実行した場合は、保護対象アプリケーションの二重起動が発生しないように配慮されています。多くのリカバリキットでは、既に起動中のアプリケーションの存在を確認した場合、特に何もせずに正常終了します。GenericARK リソースにおいても、”lkstop -f”の運用が必要であるならば、製品の ARK と同様に、二重起動への配慮が必要です。

起動状態の確認は quickCheck と同じ監視用コマンド使用されることをお勧めします。状況によって監視方式を変更すると、「restore には成功するのに、直後の quickCheck ではエラーとなる」というような問題の原因となり、切り分けが困難になる場合があります。

4-2. 起動処理および起動後の状態確認

保護対象の起動コマンドを実行し、アプリケーションの起動を行います。また、起動後に正しく起動出来たかどうかを restore スクリプトの中で確認し、起動の成功、失敗を明確にすることをお勧めします。

なお、起動コマンドの失敗や、起動後の確認において失敗が検知された場合は、不自然な状態でプロセスなどが残っている可能性も考えられます。問題とならないよう、停止処理を実行するような配慮が必要です。

4-3. スクリプトの戻り値の作成

起動コマンドの戻り値をそのまま restore スクリプトの戻り値とするのではなく、起動コマンドの戻り値を確認した上で、改めてスクリプト自体の戻り値として、0(正常)/1(失敗)のいずれかの値を返すように作成してください。以下は戻り値作成のサンプルです。

サンプルコード 2：戻り値の作成

```
# パラメータ設定

START_SUCCESS_CODE=0 # 動作確認コマンドの起動状態の戻り値
STOP_SUCCESS_CODE=0  # 動作確認コマンドの停止状態の戻り値

# 起動処理
APP_start( )
{
        logcmd LK_INFO 2 $APP_START $APP_START_ARGS

}

# 確認処理
# 引数: チェック成功と扱う値
# 戻り値: 成功時,引数と同じ値。失敗時,監視コマンドの戻り値。
APP_check( )
{
        local S_CODE=$1
        local RET=1
        for (( CNT=0 ; $CNT < ${#APP_CHECK[@]} ; CNT=$CNT+1 )) ;
                do
                if [ -x "${APP_CHECK[$CNT]}" ]; then
                        ${APP_CHECK[$CNT]} ${APP_CHECK_ARGS[$CNT]}  >>
/dev/null 2>&1 &
                        wait $!
                        RET=$?
                else
                        RET=255 # コマンドがなければエラー
                fi

                case "$RET" in
                        $S_CODE） ;; # 何もせずに次のチェックへ進む
                        *） return $RET ;; # エラー
                esac
        done

        return $S_CODE;
}
```


4-4. リトライの制御

既に案内している通り、GenericARK ではスクリプトのリトライを行いません。リトライが必要である場合は、スクリプト内で処理をリトライする仕組みを作成する必要があります。上記の「サンプルコード 2：戻り値の作成」では、アプリケーションのステータス確認コマンドの戻り値に応じて、成功、失敗、リトライに処理を分けています。

4-5. restore のタイムアウト

GenericARK リソースの restore のタイムアウト設定について説明します。

4-5-1. LifeKeeper による restore のタイムアウト（v9.1.0 以降）

/etc/default/LifeKeeper ファイルに以下のパラメータを追加することで、restore のタイムアウトを設定することができます。

GENAPP_${TTAG}_RESTORE_TIMEOUT=<秒数>

$TTAG は GenericARK リソースのタグ名で、英数字以外の記号を”_”に置き換えた文字列です。0 秒を指定すると、タイムアウトしません。また、デフォルトでもタイムアウトしません。
タイムアウトが発生した場合は、LifeKeeper ログに記録され、スクリプトはエラーで終了します。SNMP TRAP やメール通知の対象イベントにはなりません。

4-5-2. 独自の実装による restore のタイムアウト

v9.0.2 までの LifeKeeper を利用している場合や、タイムアウト発生時に実行したい処理がある場合は、スクリプト内のでタイムアウトの実装が必要です。
「サンプルコード 1：タイムアウトの実装」をご参考ください。

5. 停止(remove)スクリプトの実装

一般的なアプリケーションを保護する際の、remove スクリプトの留意点について説明します。

- ❒ 停止対象のサービスが、既に停止状態である場合への配慮
- ❒ 停止処理および停止後の状態確認
- ❒ 最終的な戻り値の作成
- ❒ リトライの制御
- ❒ remove のタイムアウト
- ❒ remove の失敗/タイムアウトをエラー終了とするかどうかの検討

5-1. 停止対象のサービスが、既に停止状態である場合への配慮

アプリケーションが停止している状態で、重ねて停止コマンドを実行するとエラーを返すアプリケーションもあります。そのため、remove スクリプト内で、停止コマンドを実行する前に、アプリケーションの状態を確認することをお勧めします。対象のアプリケーションが停止状態である場合は、正常終了として扱ってください。

5-2. 停止処理および停止後の状態確認

停止コマンドを実行し、アプリケーションの停止を行います。ここでは、以下の点について検討する必要があります。

1. 停止が正常に行われなかった場合、そのアプリケーションには強制停止の方法があるか
2. 停止コマンドを実行したあと、正常に停止したかどうかの確認は必要か

1 については、そのアプリケーションが強制停止の方法を提供しているかどうか(または KILL などによる停止をサポートしているか)に依存します。2 についても、アプリケーションの動作に依存します。停止コマンド後の確認が必要ないのであれば、実質的に remove スクリプトの失敗(停止失敗)は発生しないため、常に正常終了とすることが出来ます。

5-3. 最終的な戻り値の作成

前述の「4-3. スクリプトの戻り値の作成」でも案内していますが、停止処理についても、remove スクリプト自体の戻り値として 0(成功)か 1(失敗)のいずれかを返すよう作りこむ必要があります。

5-4. リトライの制御

停止処理においては、停止のリトライよりも強制停止などの手法を採用するケースが多いため、リトライ制御が必要となるケースは少ないかも知れません。停止コマンドの戻り値において、リトライの余地がある状況が存在するのであれば、リトライの仕組みも有効と言えます。

5-5. removeのタイムアウト

GenericARK リソースの remove のタイムアウト設定について説明します。

remove スクリプトについては、タイムアウトとなった後の制御についても、検討が必要な事項と言えます。この点については、「5-6. remove の失敗/タイムアウトをエラー終了とするかどうかの検討」を参照してください。

remove のタイムアウトに関連する機能として、リソース障害に起因するフェイルオーバーにおいて、障害を検知したノードへの停止処理要求のタイムアウトがあります。この機能の詳細と対処については、「8-1. リモート要求のタイムアウトについて」を参照してください。

5-5-1. LifeKeeperによるremoveのタイムアウト（v9.1.0以降）

/etc/default/LifeKeeper ファイルに以下のパラメータを追加することで、remove のタイムアウトを設定することができます。

GENAPP_${TTAG}_REMOVE_TIMEOUT=<秒数>

$TTAG は GenericARK リソースのタグ名で、英数字以外の記号を"_"に置き換えた文字列です。0 秒を指定すると、タイムアウトしません。また、デフォルトでもタイムアウトしません。
タイムアウトが発生した場合は、LifeKeeper ログに記録され、スクリプトはエラーで終了します。SNMP TRAP やメール通知の対象イベントにはなりません。

5-5-2. 独自の実装によるremoveのタイムアウト

v9.0.2 までの LifeKeeper を利用している場合や、タイムアウト発生時に実行したい処理がある場合は、スクリプト内のでタイムアウトの実装が必要です。
「サンプルコード 1：タイムアウトの実装」をご参考ください。

5-6. removeの失敗/タイムアウトをエラー終了とするかどうかの検討

remove は対象アプリケーションの停止処理であるため、停止の成功/失敗は、サービスの継続性に対しては大きな影響を持ちません。ただし、リソース障害検知によるフェイルオーバーの場合は、先に稼動系ノードでのリソース停止が行われます。リソースの停止処理に失敗した場合は、稼動系ノードで OS の強制再起動が行われます。
なお、本設定はコマンドラインから強制再起動を行わない設定にすることも可能です。保護するアプリケーションに合わせ、お客様で設定を行ってください。
詳細な動作、設定の変更方法に関しては、以下の URL をご参照ください。

LifeKeeper for Linux v7.0 までと v7.1 以降のリソース障害時のリカバリー動作について
http://lk.sios.com/?p=902

6. 監視(quickCheck)スクリプトの実装

quickCheck スクリプトにおける、一般的な構造と留意点について説明します。このスクリプトは、GenericARK リソースの作成において必須のものではありません。

- ❒ LifeKeeper 上のステータスの確認
- ❒ 監視コマンドの実行
- ❒ 最終的な戻り値の作成
- ❒ リトライの検討
- ❒ quickCheck のタイムアウト

6-1. LifeKeeper 上のステータスの確認

LifeKeeper では、リソース監視は常に稼動中のリソース(LifeKeeper 上のステータスが ISP であるリソース)に対して行われます。LifeKeeper は待機状態のリソース(OSU)に対しては、監視を行いません。

quickCheck を実行する際に、最初に確認しなければならないのは、このリソースは監視する必要性があるのかという点です。これを確認するためには、LifeKeeper の”ins_list”コマンドを使用します。以下は記述例です。

サンプルコード 3：リソースのステータス確認

```
STAT=$(/opt/LifeKeeper/bin/ins_list -t $TAG | cut -f7 $'-d¥001')
if [ $STAT != ISP ]
then
        log LK_INFO 7 "resource $TAG is not ISP status"
        exit 0
fi
```

APP1 リソースのステータスを確認しています。ins_list コマンドは、リソースの詳細情報を画面に出力します。コマンド実行時のリソースのステータスは、7 番目のフィールドに格納されていますので、cut コマンドでその部分を切り出してください。なお、ins_list コマンドは、各情報のデリミタとして[^A]のメタ・キャラクタを使用しています([ctrl] + [v] , [ctrl] + [a])。

※ 上記処理はremoveとquickCheckでの競合状態を回避する為に必要な処理です。リソースに対して remove が実行されるとステータスが ISU に変更されます。その為、ステータスが ISP でなければ quickCheck を中断して remove 実行中のリソースに対して quickCheck を行い誤った障害検出を防ぎます。

6-2. 監視コマンドの実行

アプリケーションが監視用のコマンドを用意しているのであれば、そのコマンドを実行してアプリケーションの状態を確認してください。また、一般的に状態確認に使用できるコマンドについて、以下に例をご紹介します。

コマンド例

- プロセスの存在確認
  # ps auxwww | grep <プロセス名> | grep -v grep
- プロセスおよび待ち受けポートの確認
  # netstat -anp | grep <port 番号やプロセス名など>
- プロセスが必要なファイルを開いているかの確認
  # lsof | grep <プロセス名など>
- ポートの状態確認
  # nmap -p <ポート番号> <仮想 IP アドレス>
  # nc <仮想 IP アドレス> <ポート番号>

※　lsof は、open しているファイルが多い場合はシステムに負荷を与えることがあります。

6-3.　最終的な戻り値の作成

監視処理についても、スクリプト自体の戻り値として 0(成功)か 1(失敗)のいずれかを返すように作りこんでください。

6-4.　リトライの検討

監視コマンドの実行結果が正常以外であった場合は、そのステータスに応じてリトライを行うケースも考えられますが、リトライを行った分だけ障害を判定するまでの時間が余計にかかることにもなります。監視コマンドの実行結果にて、リトライによる確認が必要な応答があった場合のみリトライを行うなどの検討が必要です。

6-5.　quickCheck のタイムアウト

GenericARK リソースの quickCheck のタイムアウト設定について説明します。

6-5-1. LifeKeeper による quickCheck のタイムアウト（v9.0.2 以前）

/etc/default/LifeKeeper ファイルに以下のパラメータを追加することで、
quickCheck のタイムアウトを設定することが出来ます。

${TTAG}_TIMEOUT=<秒数>

$TTAG は GenericARK リソースのタグ名で、英数字以外の記号を"_"に置き換えた文字列です。
スクリプト内で実行した確認コマンドの応答が返らないといった場合、quickCheck スクリプトは応答待ち状態となります。quickCheck の実行から、上記のパラメータで指定した秒数が経過した時点で、quickCheck スクリプトは LifeKeeper により kill されます。

このタイムアウトは、明確にエラーを検知した訳ではないため、正常終了として扱われます。そのため、フェイルオーバーのトリガーとはなりません。

なお、タイムアウトの設定を行わなかった場合は、GenericARK の quickCheck には 22 秒のタイムアウトが適用されます。

また、このタイムアウトの発生の記録は、LifeKeeper のログ中に出力されるのみであり、SNMP TRAP やメール通知の対象イベントにはなりません。

6-5-2. LifeKeeper による restore のタイムアウト（v9.1.0 以降）

/etc/default/LifeKeeper ファイルに以下のパラメータを追加することで、restore のタイムアウトを設定することができます。

GENAPP_${TTAG}_QUICKCHECK_TIMEOUT=<秒数>

$TTAG は GenericARK リソースのタグ名で、英数字以外の記号を"_"に置き換えた文字列です。0 秒を指定すると、タイムアウトしません。また、デフォルトでもタイムアウトしません。
タイムアウトが発生した場合は、LifeKeeper ログに記録され、SNMP TRAP やメール通知の対象イベントにはなりません。また、このタイムアウトは明確にエラーを検知した訳ではないため、正常終了として扱われます。そのため、フェイルオーバーのトリガーとはなりません。

6-5-3. 独自の実装による quickCheck のタイムアウト

監視のタイムアウト発生をフェイルオーバーのトリガーとする場合、またはタイムアウト発生時に実行したい処理があるといった場合は、LifeKeeper が用意する監視の仕組みではなく、restore や remove の項で案内してきたような、スクリプト内でのタイムアウトの実装が必要です。(3-1 サンプルコード 1 をご参照ください)

v9.0.2 までの LifeKeeper で独自にタイムアウトの仕組みを作成した場合は、「6-5-1. LifeKeeper による quickCheck のタイムアウト（v9.0.2 以前）」で紹介した、LifeKeeper が標準的に用意しているタイムアウト設定との共存が必要になります。
具体的な方法としては、/etc/default/LifeKeeper ファイルの"${$TAG}_TIMEOUT=<秒数>"には、スクリプト内で定義したタイムアウトよりも、長い秒数を設定してください。

また、"${TTAG}_TIMEOUT=<秒数>"（v9.0.2 以前）や"GENAPP_${TTAG}_QUICKCHECK_TIMEOUT=<秒数>"（v9.1.0 以降）として設定するタイムアウトの秒数は、/etc/default/LifeKeeper の LKCHECKINTERVAL パラメータで定義された値よりも、小さい値をセットしてください。
LKCHECKINTEVAL は、LifeKeeper に登録されたリソース全体の監視サイクルであり、初期値は 120 秒です。個々のリソースのタイムアウト値は、このパラメータが示す値よりも小さくなければなりません。
タイムアウトの値が LKCHECKINTERVAL パラメータで定義された値よりも大きな場合は、監視処理が時間内に完了しなかった旨のログが出力されます。

6-5-4. quickCheck のタイムアウト後の取り扱い

quickCheck のタイムアウトを独自に実装した場合は、タイムアウトが発生した後の動作についての検討も必要です。以下は検討事項の例となります。

- タイムアウトの発生をエラーとする場合

タイムアウトが発生した場合は、戻り値が 1(失敗)になるように設定し、recover スクリプトでフェイルオーバー処理の実行や、ノードの再起動の実行を記述する方法が有効です。

- タイムアウトの発生をエラーとはしないが、タイムアウトが発生したという情報は必要な場合

タイムアウトが発生した場合に呼び出される処理において、メール送信や SNMPTRAP の実行を記述する方法が有効です。以下はタイムアウト時に管理者にメールを送信する場合のサンプルです。

サンプルコード 4：タイムアウト発生時のメール送信

```
handlealarm()
{
        LANG=C; date; echo "Script was hung. Forcibly terminating."

        # INT を受け取った後、mail コマンドで管理者宛にタイムアウトが発生した旨のメールを
        # 送信します。$TO_MESSAGE には、管理者に送るメールの本文を格納してください。
         echo $TO_MESSAGE | mail -s"$(date -R) timeout occurred in ¥
                                quickCheck for $TAG" <メールアドレス>

        if [ "$(ps $$ > /dev/null 2>&1; echo $?)" -ne 1 ]
        then
                kill -TERM $$
                if [ "$(ps $$ > /dev/null 2>&1; echo $?)" -ne 1 ]
                then
                     LANG=C; date; echo "Script was still hung. sending KILL signal"
                     kill -KILL $$
                fi
        fi
        exit 0
}
```


7. 再起動(recover)スクリプトの実装

recover スクリプトは、quickCheck スクリプトが戻り値 1 で終了した後、ローカルノードでサービスの再起動を行い、復旧を試みる仕組みです。このスクリプトは、GenericARK リソースの作成において必須のものではありません。

ここでは、recover を作成する場合の留意点について説明します。

- ❒ recover の有無についての検討
- ❒ 制御コマンドの実行
- ❒ 最終的な戻り値の作成
- ❒ リトライの検討
- ❒ recover のタイムアウト

7-1 recover の有無についての検討

- recover スクリプトを作成しない場合は、quickCheck スクリプトが戻り値 1 (失敗)で終了した後は、フェイルオーバー処理が実行されます。
- recover スクリプトを作成した場合は、スクリプトに記載されている動作を行います。

recover スクリプトの実行でサービスの回復が見込める場合は、一般的にフェイルオーバーを行った場合よりもサービスの回復時間が短くなります。しかし、recover では回復しない場合は、他のノードへ切り替えてしまった方が早期に復旧するといったケースも考えられます。

サービスの復旧にはどちらがより速いのかなどを検討の上、recover を作成するかを決定してください。

7-2 制御コマンドの実行

アプリケーション側に再起動用の制御コマンドが用意されているのであれば、それを実行するのが最も簡単な手法となります。そのようなコマンドが用意されていないアプリケーションに対しては、スクリプトの中で再起動の仕組みを作成しなければなりません。

LifeKeeper では、殆どの ARK が recover を用意しています。recover の内容は ARK によって様々ですが、いわゆるサーバアプリケーション(DB やメール、ファイル共有など)においては、以下のステップで再起動を試みます。

1. アプリケーションの状態確認(quickCheck と同等)
2. アプリケーションの停止(remove と同等)
3. アプリケーションの起動(restore と同等)
4. 再起動後のアプリケーションの状態確認(quickCheck と同等)

上記の 4 の状態確認において、復旧が確認できない場合は、待機ノードへのフェイルオーバーを行います。

7-3. 最終的な戻り値の作成

これまでに説明して参りました他のスクリプトと同様に、スクリプト自体の戻り値を用意する必要があります。また、停止や起動コマンドそのものの失敗などがあった場合、その時点で失敗終了として戻り値 1(失敗)を返すといった制御とすることで、recover の失敗を早期に確定し、迅速に待機ノードへのフェイルオーバーを開始するといった制御も可能です。

7-4. リトライの検討

他のスクリプトと同じく、GenericARK には recover のリトライを行う仕組みはありません。リトライが必要である場合は、スクリプトの内部でリトライの仕組みを作成する必要があります。

7-5. recover のタイムアウト

GenericARK リソースの recover のタイムアウト設定について説明します。
タイムアウト後の扱いについては、失敗扱いとするのが妥当と考えられます。

7-5-1. LifeKeeper による recover のタイムアウト（v9.1.0 以降）

/etc/default/LifeKeeper ファイルに以下のパラメータを追加することで、recover のタイムアウトを設定することができます。

```
GENAPP_${TTAG}_RECOVER_TIMEOUT=<秒数>
```

$TTAG は GenericARK リソースのタグ名で、英数字以外の記号を”_”に置き換えた文字列です。0 秒を指定すると、タイムアウトしません。また、デフォルトでもタイムアウトしません。
タイムアウトが発生した場合は、LifeKeeper ログに記録され、スクリプトはエラーで終了します。SNMP TRAP やメール通知の対象イベントにはなりません。

8. その他の留意事項

restore、remove、quickCheck、recover の各スクリプトに特有の説明は以上です。下記に、その他の留意事項について紹介します。

8-1. リモート要求のタイムアウトについて

LifeKeeper はフェイルオーバーの際に、フェイルオーバー先のノードから、フェイルオーバー元(障害発生ノード)に対して、リソースの停止を要求します。この要求には、応答時間のタイムアウトが設けられています。このタイムアウトの仕組みについて以下のリソース階層を例として説明します。

リソース階層	Server1 priority	Server2 priority
Generic APP	1	10
Virtual IP (障害部位)	1	10
Filesystem	1	10
Device	1	10
Disk	1	10

1. 上記の例では、Virtual IP にて障害を検知しています。この後、Virtual IP リソースはローカルリカバリに失敗し、Server1 から Server2 にフェイルオーバーを行うという動作を想定します。

2. 障害となった Virtual IP は、Generic APP リソースを頂点としたリソースツリーに含まれています。そのため、フェイルオーバーの対象となるリソースは Generic APP リソースとなり、その下位にある各リソースは、Generic APP リソースの起動に必要であるという理由で、フェイルオーバーの対象になります。

3. Virtual IP リソースが障害となった後、Server2 では、頂点にある Generic APP が起動できるかを確認します(まだ実際の restore スクリプトは実行しません)。この時点で、Generic APP リソースの起動には、Virtual IP～Disk までの各リソースの起動が必要であることを認識します。

4. しかし、障害部位である Virtual IP 以下の Filesystem～Disk の各リソースは、この時点では Server1 側で起動状態(ISP ステータス)のままです。そのため、Server2 は、Server1 に対して、Disk リソースの停止を命じます。

5. Disk リソースを停止するためには、Server1 は Disk リソースに依存する上位の全リソースを停止しなければなりません。全てのリソースの停止が完了すると、Server1 は停止が完了したことを Server2 に通知します。停止完了の連絡を受けた Server2 は、各リソースを下位のものから順番に起動を行います。Generic APP の起動が完了すれば、フェイルオーバーは成功です。

この一連の流れの中で、項番 4 にある Server2 から Server 1 への停止要求の実行から、5 の Server1 から Server2 への停止要求の完了通知までの動作に対して、リモートノードに対する要求のタイムアウトがセットされています。

これは/etc/default/LifeKeeper に、REMOTETIMEOUT パラメータを用いて定義されています。このパラメータの初期値は 900 秒にセットされており、4 の停止要求の発行から 900 秒が経過した時点で、「リモートからの停止要求がタイムアウトした」と判定されます。リモートからの停止要求がタイムアウトした場合は、フェイルオーバーの失敗と判定されます。

remove のタイムアウトをスクリプト内に作成する場合は、この REMOTETIMEOUT によるタイムアウト時間も考慮する必要があります。停止に長い時間がかかるようなアプリケーションの場合は、remove 内のタイムアウトとともに、REMOTETIMEOUT の調整も検討しなければなりません。

8-2 ログの出力方法について

LifeKeeper for Linux v8.0 より、ログシステムが LifeKeeper for Linux 独自のバイナリ形式のログから、Linux OS において標準的な syslog 機能を通して行われるようになりました。変更に伴い LifeKeeper のログは、以下の関数を使用して出力させます。

- logmsg 関数
  ログのみへ出力します。

- plogmsg 関数
  ログへの出力のほか、perform_action コマンドや GUI からのリソース起動および 停止操作時のメッセージとしても出力されます。

詳細につきましては、以下の URL をご参考ください。

[Linux]LifeKeeper for Linux のログシステム変更に伴うログ出力の変更について
http://lk.sios.com/?p=2078

9.免責事項

- 本書に記載された情報は予告なしに変更、削除される場合があります。最新のものをご確認ください。
- 本書に記載された情報は、全て慎重に作成され、記載されていますが、本書をもって、その妥当性や正確性についていかなる種類の保証もするものではありません。
- 本書に含まれた誤りに起因して、本書の利用者に生じた損害については、サイオステクノロジー株式会社は一切の責任を負うものではありません。
- 第三者による本書の記載事項の変更、削除、ホームページ及び本書等に対する不正なアクセス、その他第三者の行為により本書の利用者に生じた一切の損害について、サイオステクノロジー株式会社は一切の責任を負うものではありません。
- システム障害などの原因によりメールフォームからのお問い合せが届かず、または延着する場合がありますので、あらかじめご了承ください。お問い合せの不着及び延着に関し、サイオステクノロジー株式会社は一切の責任を負うものではありません。

サイオステクノロジー株式会社
住所 ： 〒106-0047
東京都港区南麻布 2 丁目 12-3　サイオスビル
URL：http://www.sios.com

