3.28 基幹系システムにパッケージソフトを適用する際の教訓（その 2）（T28）

教訓 T28　パッケージを更新するときは、変更内容の詳細確認と回帰テストで二重に安全を確保せよ

問題

A 社で保有する社内特定業務向け基幹業務システムは、現場部門に対して 24 時間のオンライン業務サポートを実施している。業務システムは当該用途向けのパッケージを購入して構築されており、アプリケーションサーバ、DB サーバ、他システムとの連携サーバなど各層のサーバは冗長構成になっている。構築したシステムは本稼働後、2 年間安定稼働したのち、2 週間前にメジャーバージョンアップを実施したばかりであった。

ある日、アプリケーションサーバからの反応が突然なくなった。そこから、システム回復までの経緯はおおよそ以下のとおりである。

7:48　サービスが使用できない（応答がない）と現場から多数の連絡が入るようになった
すぐにアプリケーションサーバやデータ連携用のサーバを再起動したが、状況は全く改善せず

8:51　未処理のまま滞留している他システムからの連携データを一度すべて削除し、他システムとのデータ連携も一時的に停止した上でアプリケーションサーバを再起動した結果、レスポンスが回復

10:30　正常の構成に戻してサービスを再開

問題発生後、サービス再開まで 3 時間を要した間、当該特定業務の運用が完全に停止し、当該業務を通してエンドユーザに提供しているサービスにも大きな影響が出た結果、その復旧を含めて A 社のビジネスに大きな損害をもたらした。ただし、当サービスの不測の停止に備えて、このサービスを使用する運用現場とサービスが稼働しない状態でも実施できる範囲の業務対応を準備していたことから、最も影響のある業務の停止だけは回避できた。

原因

アプリケーションサーバがダウンした直接の原因は、2 週間前にパッケージをバージョンアップした際に置き替えられたアプリケーションに混入していたバグであった。パッケージの開発元ではメジャーバージョンアップの際に、一部のアプリケーションについて性能改善のための改造（アプリサーバのキャッシュで更新制御）を実施していたが、その際、誤って複数の処理間でのリソース競合が発生す

るような内容での修正を実施したことにより互いに処理待ちになっていたことが，サーバが反応を返さなくなった原因であった。

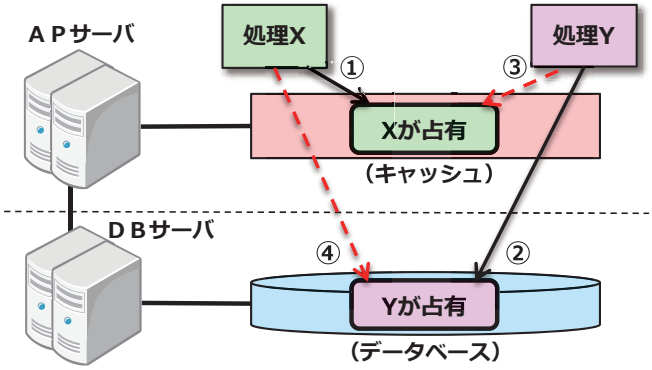

図 3.28－1　サーバ資源占有競合の概要

上図において処理 X はキャッシュ、ディスクの順で資源を確保し、処理 Y はその逆の順で資源を確保する不統一な作りになっていた。その上、それぞれが自分の占有した資源を開放せずに他方が占有した資源の解放を待つ作りになっていたため、両者ともに相手の資源開放待ちで停止した。

また、パッケージ開発元から提供されるリリースノートには、機能レベルの大きな変更までしか記載されておらず、今回のような個々の処理に対する性能改善レベルの修正が実施されていることは A 社には伝わっていなかった。そのため、A 社ではその修正内容の是非やリスクの検討、テスト環境での事前の動作確認はできていなかった。

対策

パッケージ開発元から発生原因が上記である旨の報告を受けた A 社では、パッケージの当該個所の修正だけでなく、今後もし同様のトラブルが再発したときの備えとして、パッケージ開発元と協議の上で、以下の対策を実施した。

【対策 1】パッケージアップデートに含まれるすべての修正に対するレビューの設定

A 社では、これ以降、パッケージ開発元からアップデートが提供される際に、アップデートに含まれるすべての修正に対して内容の詳細な報告を求め、両者による修正内容のレビュー会を実施することをパッケージ開発元に承諾させ、提示されたすべての修正内容に応じた受入テストを計画することとした。また、A 社内ではレビュー会を実施していないアップデートの運用環境への適用を一切禁止するルールを敷いた。これにより、パッケージ開発元から報告された修正に対しては運用環境に適用する前に A 社でも内容を認識し、確認することができるようになった。

また、レビュー会においては、今回の障害の直接原因になった資源占有（排他）順序の不統一のような誤りを、パッケージ開発元で出荷前にちゃんと摘出できるようなプロセス改善が適切に計画され、実施されているかどうかをあわせてチェックすることにより、不具合混入の再発防止を行った。

【対策 2】回帰テストシナリオの作成とテストの実施

A 社では、さらにパッケージ開発元の管理者でも認識していないような修正が、開発元内での修正報告漏れなどにより実施されているような場合に備え、回帰（リグレッション）テスト用のテストシナリオを作成して、パッケージが入替え前後で同一の動作をすることを確認することを手順化した。さらに、このテストシナリオを自動で動作させられるテスト環境を構築し、システムを更新するときに回帰テストを実施する際の負荷軽減とテスト手順の誤りによる障害摘出漏れの防止を実現した。

【対策 3】アプリケーションレベルでの死活監視の組込み

A 社では、今回のアプリケーションサーバと DB サーバの間での資源競合のような、サーバ単体でのデッドロック監視では検出しきれないエラーへの対策として、アプリケーションレベルでの死活監視を新たにシステムに加えた。さらに、実行中のプログラムがエラー発生を検知したら、すぐにシステム運用担当者に直接メールが発信されるようにアプリケーションに機能を追加し、システムに組み込んだ。これにより、場合によってはエラー発生時の初動を、サービスの利用者からの連絡よりも早く開始することができるようになった。

効果

上記の施策により、この事例では以下の効果を得た。

(1) アップデート適用前の修正内容の全件把握とテストによる確認の確実な実施

提供されたアップデートを運用環境に適用しても問題ないかどうかの確認を、これまでよりも正確かつ緻密に実施できるようになった。また、アップデート受入れ時の動作確認においても、修正個所に直接関係するテスト項目の設定を高精度で実施できるようになり、確認の精度が向上した。

(2) パッケージアップデート前後での同一動作の担保

パッケージをアップデートしても、アップデートに関係しない部分の動作がこれまでと同様であることを維持できているかどうかを、より低負荷かつ確実に確認できるようになった。

教訓

この事例からは、パッケージのように他社から実行形式モジュールだけを提供され、その内容についてまでは利用者側では詳細を知り得ないソフトウェアを業務システムに組み込んでいるシステムを更新する際に考慮すべき、以下の注意点が教訓として活用できると考えられる。

(1) アップデートを適用する際には、修正内容を詳細まで把握すること

通常、パッケージソフトウェアの更新モジュールにはリリースノートが添付されているが、そこに記載される内容は、業務上の機能レベルでの追加や変更が大まかに記載されていることがほとんどである。パッケージの個々のモジュールのどこにどのような修正をしたかについてまでは、リリースノートに記載されていないどころか、開発元でも詳細レベルまでは管理しきれていないのが実態ではないかと思われる。パッケージの利用者はそれらの不十分な情報をもとに、自己の運用するシステムに適用する前に受入確認を行い、その後に運用環境に適用する。今回の事例のような高度の運用品質を求められる

システムにパッケージを適用する必要がある場合には、特別のサポートにより修正内容を詳細に報告させ、それに基づいた厳密な受入れテストを実施できるような体制を採ることが必要である。

(2) 回帰テストの実施により、予期しない変更に備えること

すべて報告すると契約していても、漏れが生ずるリスクは当然残る。そのリスクの顕在化に備えて、パッケージがアップデート前後で同じ動作をすることを、上記 (1) の受入れテストとは別に、回帰テストによって確認することが望ましい。できればすべての機能について、あるいは、それが現実的には困難な場合には、パッケージの主要な機能についてテストシナリオを用意して、それをシステム更新前に必ず実行するように準備しておけば、アプリケーションの入替えだけでなく、OS やミドルウェアのアップデートや、機器の入替えの際にも動作確認に使用できることから、確認の範囲が広がり、システム保守の信頼性をより向上させることができる。

システム構築において、OS、ミドルウェア、アプリケーションの各層にパッケージ製品を組み込むことは最早当たり前になってきている。パッケージを適用してシステムを構築した場合、そのアップデートが出ても自己の運用するシステムに影響がある修正でない限り適用をしないユーザも多い。ただし、自己のシステムに影響があるアップデートや、あるアップデートを適用する際に、これまで適用を保留してきた大量のアップデートをまとめて適用することが必要になった場合、今回の事例のようなアプローチによって、少しでも安全にアップデートを行うことが必要になる。また、もしパッケージに選択の余地があるなら、アップデートの提供時により詳細な内容の更新情報を提供するパッケージ開発元を選択することも必要になってくると考えられる。この教訓では、システムに組み込まれているパッケージにアップデートを実施する際には、適用するシステムに求められる運用品質に応じた更新情報の取得努力と、それが得られない場合のリスクヘッジを用意することが重要であることを発信したい。

教訓タイトルは、「パッケージを更新するときは、変更内容の詳細確認と回帰テストで二重に安全を確保せよ」とした。

なお、ソフトウェアパッケージと同等にブラックボックス化しやすいハードウェアコンポーネントを交換した際に発生したシステム障害の事例として、教訓 T12「新製品は、旧製品と同一仕様と言われても、必ず差異を確認！」もあわせて参照を願う。